DSP LAB MANUAL

For

5TH SEMESTER ELECTRONICS AND COMMUNICATION ENGINEERING

SITECH ELECTRONICS
No.2,"ASHRAYA APARTMENTS"
Opp.to S B I, Athmananda Colony
Sulthanpalya, R.T.Nagar, BENGALURU- 560032,
PH: +91 80-23339222, Tele /Fax: 23335449
E-Mail: info@chipmaxindia.com
Web: www.chipmaxindia.com

CONTENTS

PART-A

EXPERIMENT NO-1

AIM: VERIFICATION OF SAMPLING THEOREM

Sampling: Is the process of converting a continuous time signal into a discrete time signal. It is the first step in conversion from analog signal to digital signal.

Sampling theorem: Sampling theorem states that "Exact reconstruction of a continuous time base-band signal from its samples is possible, if the signal is band-limited and the sampling frequency is greater than twice the signal bandwidth".
i.e. $f_s > 2W$, where W is the signal bandwidth.

Nyquist Rate Sampling: The Nyquist rate is the minimum sampling rate required to avoid aliasing, equal to the highest modulating frequency contained within the signal. In other words, Nyquist rate is equal to two sided bandwidth of the signal (Upper and lower sidebands)
To avoid aliasing, the sampling rate must exceed the Nyquist rate. i.e. $f_s > f_N$

Program:

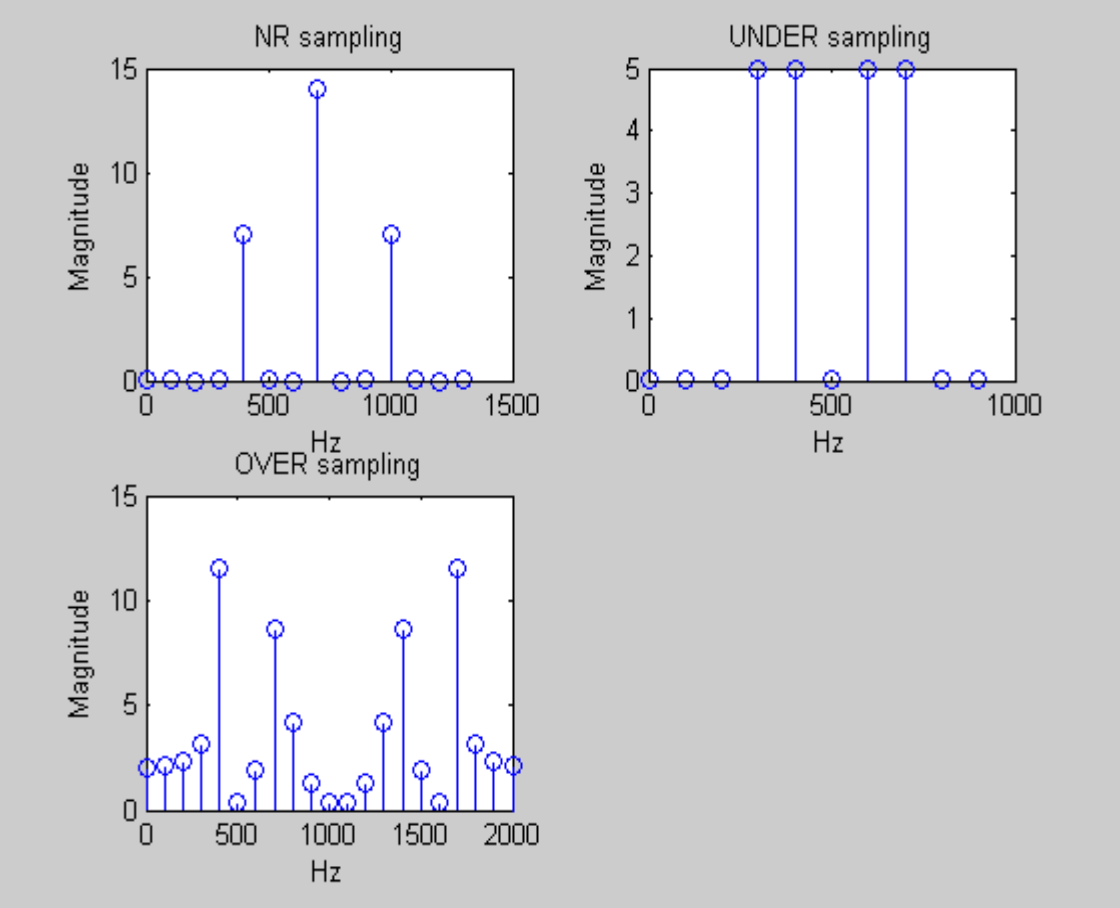
```
% Nyquist Rate Sampling.
clc;                          % Clear screen
fs = 1400;                    % Sampling frequency, fs = 1400Hz
t = 0:1/fs:13/fs;             % Number of samples
x = cos(2*pi*400*t)+cos(2*pi*700*t); % Input signal
xm = abs (fft(x));            % Determine the absolute FFT of the input signal
disp('xm');                   % Displays xm on command window
disp (xm);                    % Displays values of xm on command window
k = 0:length(xm)-1;           % Number of  samples to be plot on x-axis
subplot (2,2,1);              % Divide the figure window to plot the output
stem (100*k, xm);             % Plot the output
xlabel ('Hz');                % Name x-axis as "Hz"
ylabel ('Magnitude');         % Name y-axis as "Magnitude"
title ('NR sampling');        % Title is "NR Sampling"

%UNDER Sampling.
fs = 1000;                    % Sampling frequency, fs = 1000Hz
t = 0:1/fs:9/fs;              % Number of samples
x = cos(2*pi*400*t)+cos(2*pi*700*t); % Input signal
xm1 = abs(fft(x));            % Determine the absolute FFT of the input signal
disp('xm1');                  % Displays xm1 on command window
disp (xm1);                   % Displays values of xm1 on command window
k = 0:length(xm1)-1;          % Number of  samples to be plot on x-axis
subplot(2,2,2);               % Divide the figure window to plot the output
stem(100*k, xm1);             % Plot the output
xlabel('Hz');                 % Name x-axis as "Hz"
ylabel('Magnitude');          % Name y-axis as "Magnitude"
```

```
title('UNDER sampling');      % Title is "UNDER Sampling"

%OVER Sampling.
fs = 2000;                    % Sampling frequency, fs = 2000Hz
t = 0:1/fs:20/fs;             % Number of samples
x = cos(2*pi*400*t)+cos(2*pi*700*t); % Input signal
xm2 = abs(fft(x));            % Determine the absolute FFT of the input signal
disp('xm2');                  % Displays xm2 on command window
disp (xm2);                   % Displays values of xm2 on command window
k = 0:length(xm2)-1;          % Number of  samples to be plot on x-axis
subplot(2,2,3);               % Divide the figure window to plot the output
stem(100*k,xm2);              % Plot the output
xlabel('Hz');                 % Name x-axis as "Hz"
ylabel('Magnitude');          % Name y-axis as "Magnitude"
title('OVER sampling');       % Title is "OVER Sampling"
```

Output :

```
xm
  Columns 1 through 7
    0.0000    0.0000    0.0000    0.0000    7.0000    0.0000    0.0000

  Columns 8 through 14
   14.0000    0.0000    0.0000    7.0000    0.0000    0.0000    0.0000

xm1
  Columns 1 through 7
    0.0000    0.0000    0.0000    5.0000    5.0000    0.0000    5.0000

  Columns 8 through 10
    5.0000    0.0000    0.0000

xm2
  Columns 1 through 7
    2.0000    2.0741    2.3496    3.1607   11.5127    0.4052    1.8998

  Columns 8 through 14
    8.6165    4.2355    1.2552    0.3357    0.3357    1.2552    4.2355

  Columns 15 through 21
    8.6165    1.8998    0.4052   11.5127    3.1607    2.3496    2.0741
```

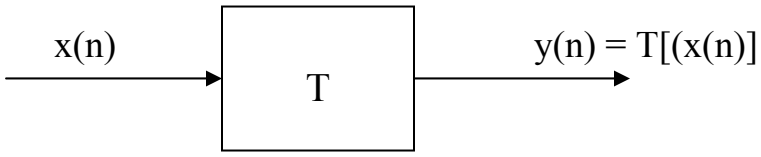
Graph:


NR sampling
Magnitude
Hz
0
5
10
15
0
500
1000
1500
UNDER sampling
Magnitude
Hz
0
1
2
3
4
5
0
500
1000
OVER sampling
Magnitude
Hz
0
5
10
15
0
500
1000
1500
2000

EXPERIMENT NO-2

AIM: IMPULSE RESPONSE OF A GIVEN SYSTEM

A discrete time system performs an operation on an input signal based on a predefined criteria to produce a modified output signal. The input signal $x(n)$ is the system excitation, and $y(n)$ is the system response. The transform operation is shown as,



If the input to the system is unit impulse i.e. $x(n) = \delta(n)$ then the output of the system is known as impulse response denoted by $h(n)$ where,

$$h(n) = T[\delta(n)]$$

we know that any arbitrary sequence $x(n)$ can be represented as a weighted sum of discrete impulses. Now the system response is given by,

$$y(n) = T[x(n)] = T\left[\sum_{k=-\infty}^{\infty} x(k)\ \delta(n-k)\right] \qquad \ldots(1)$$

For linear system (1) reduces to

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)\ T[\delta(n-k)] \qquad \ldots(2)$$

The response to the shifted impulse sequence can be denoted by $h(n, k)$ is denoted by,

$$h(n, k) = T[\delta(n-k)] \qquad \ldots(3)$$

For a time-invariant system

$$h(n, k) = h(n-k) \qquad \ldots(4)$$

Using (4) in (3) we obtain,

$$T[\delta(n-k)] = h(n-k) \qquad \ldots(5)$$

Using (5) in (2) we get,

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)\ h(n-k) \qquad \ldots(6)$$

For a linear time-invariant system if the input sequence is $x(n)$ and impulse response $h(n)$ is given, we can fine output $y(n)$ by using eqn (6).
Which is known as convolution sum and can be represented by $y(n) = x(n) * h(n)$

For Example let's find out an impulse response of a difference equation.
The general form of difference equation is,

$$y(n) = \sum_{k=1}^{M} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k) \qquad \ldots(7)$$

For input $x(n) = \delta(n)$

$$y(n) = \sum_{k=0}^{M} b_k\, x(n-k) = 0 \qquad \text{for } n > M$$

Now (7) can be written as,

$$y(n) = \sum_{k=0}^{N} a_k\, y(n-k) = 0 \qquad a_0 = 1 \qquad \ldots(8)$$

The solution of eqn (8) is known as homogeneous solution. The particular solution is zero since $x(n) = 0$ for $n > 0$, that is

$$y_p(n) = 0$$

Therefore we can obtain the impulse response by solving the homogeneous equation and imposing the initial conditions to determine the arbitrary constants.

Example:
Let's take a second order difference equation

$$y(n) - (1/6)\, y(n-1) - (1/6)\, y(n-2) = x(n) \qquad \ldots(9)$$

For impulse response the particular solution $y_p(n) = 0$

So, $y(n) = y_h(n)$ $\qquad \ldots(10)$

Let $y_h(n) = \lambda^n$ $\qquad \ldots(11)$
Substituting (11) in (9) we get,

$\lambda^n - (1/6)\, \lambda^{n-1} - (1/6)\, \lambda^{n-2} = 0$ $\quad$ ($x(n) = 0$ for homogeneous solution)
or
$\lambda^2 - (1/6)\, \lambda - (1/6) = 0$ $\qquad \ldots(12)$

The roots of the characteristic equation are,
$\lambda_1 = 1/2, \lambda_2 = -1/3$

The solution $y_h(n) = C_1 (1/2)^n + C_2 (-1/3)^n$ …(13)

For impulse $x(n) = \delta(n)$; $x(n) = 0$ for $n > 0$ and $x(0) = 1$

Substituting the above relations in eqn (9) we have,

For $n = 0$,

$$y(0) – (1/6)\, y(-1) –(1/6)\, y(-2) = x(0) = 1$$

i.e. $y(0) = 1.$ …(14)

For $n = 1$,

$$y(1) – (1/6)\, y(0) –(1/6)\, y(-1) = x(1)$$
$$y(1) – (1/6) = 0$$
$$y(1) = 1/6.$$ …(15)

From eqn (13)

$$y(0) = C_1 + C_2$$ …(16)

$$y(1) = (1/2)C_1 – (1/3)C_2$$ …(17)

Comparing equations (14), (15), (16) and (17) we get,

$$C_1 + C_2 = 1$$
$$(1/2)C_1 – (1/3)C_2 = (1/6)$$

Solving for C_1 and C_2 we get,

$$C_1 = 3/5, C_2 = 2/5$$

Therefore, the solution

$$y(n) = (3/5)(1/2)^n + (2/5)(-1/3)^n$$ …(18)

Now, let's find out impulse response.
We know that,

$$y(n) = x(n) * h(n)$$
$$h(n) = y(n) / x(n)$$ …(19)

Compute $y(n)$ for various values of n
Substitute $n = 0, 1, 2, 3$ in eqn (18) we get,

$y(0) = 1.0000$
$y(1) = 0.1667$
$y(2) = 0.1944$
$y(3) = 0.0602$

So, $y(n) = \{1.0000, 0.1667, 0.1944, 0.0602\}$
We know that, $x(n) = 1$

So, the impulse response for the given difference equation using eqn (19) is
$h(n) = \{1.0000, 0.1667, 0.1944, 0.0602\}$

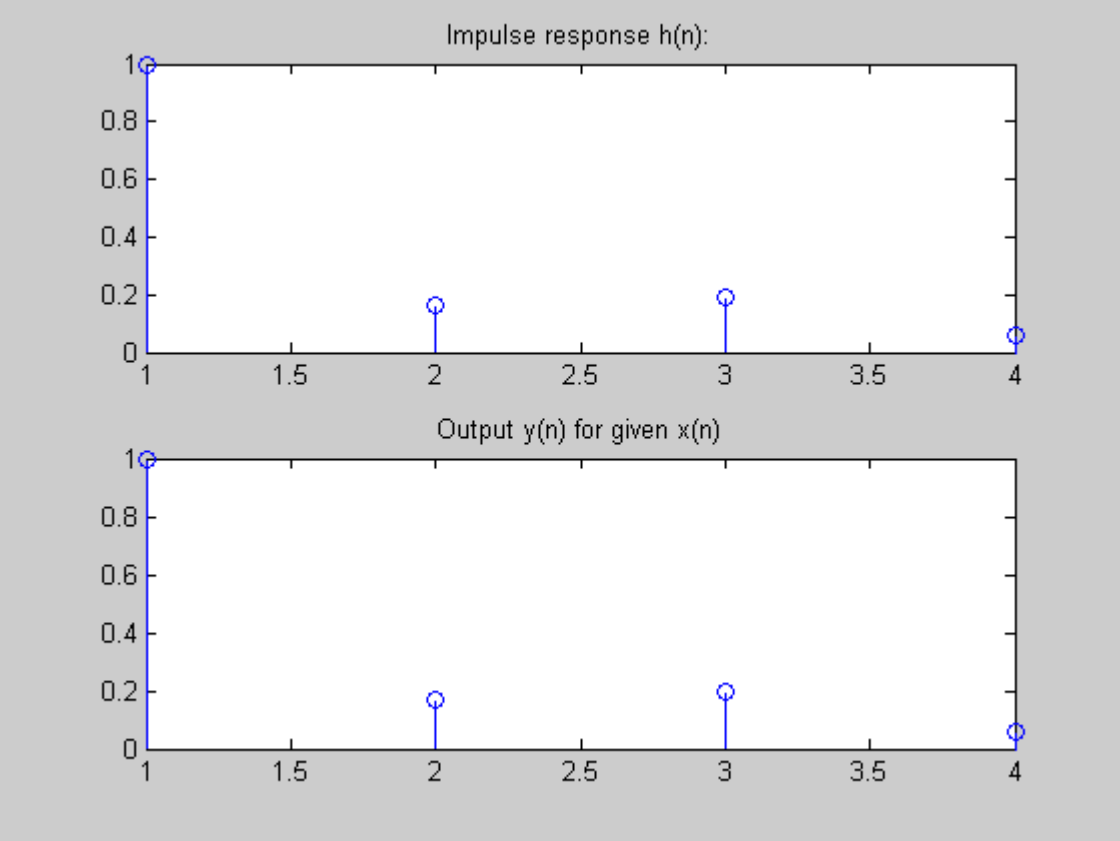
Program:

```
clc;                          % Clear screen
nr = [1];                     % Numeartor co-efficients
dr = [1,-1/6,-1/6];           % Denominator co-efficients
h = impz(nr,dr,4);            % Computes the impulse response of a given
                              % difference equation, returns the co-efficients
disp('Impulse response h(n):');% Displays impulse response
disp(h);                      % Displays impulse response on command window
subplot(2,1,1);               % Divide the figure window to plot the output
stem(h);                      % Displays the impulse response
title('Impulse response h(n):');% Title as impulse response
x = [1];                      % Input co-efficients
y = conv (h, x);              % Convolution of input and impulse co-efficients to get the
                              % output
disp('Output y(n):')          % Displays Output y(n)
disp(y);                      % Displays the output on command window
subplot(2,1,2);               % Divide the figure window to plot the output
stem(y);                      % Plots the output
title('Output y(n) for given x(n)'); % Title as "Output y(n) for given x(n)"
```

Output:

```
Impulse response:
  1.0000
  0.1667
  0.1944
  0.0602
Output y(n):
  1.0000
  0.1667
  0.1944
  0.0602
```

Graph:


Impulse response h(n):
1
0.8
0.6
0.4
0.2
0
1
1.5
2
2.5
3
3.5
4
Output y(n) for given x(n)
1
0.8
0.6
0.4
0.2
0
1
1.5
2
2.5
3
3.5
4

EXPERIMENT NO-3

AIM: TO IMPLEMENT LINEAR CONVOLUTION OF TWO GIVEN SEQUENCES

Theory: Convolution is an integral concatenation of two signals. It has many applications in numerous areas of signal processing. The most popular application is the determination of the output signal of a linear time-invariant system by convolving the input signal with the impulse response of the system.
Note that convolving two signals is equivalent to multiplying the Fourier Transform of the two signals.

Mathematic Formula:
The linear convolution of two continuous time signals $x(t)$ and $h(t)$ is defined by

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau)\, d\tau$$

For discrete time signals $x(n)$ and $h(n)$, is defined by

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$$

Where $x(n)$ is the input signal and $h(n)$ is the impulse response of the system.

In linear convolution length of output sequence is,
length(y(n)) = length(x(n)) + length(h(n)) – 1

Graphical Interpretation:

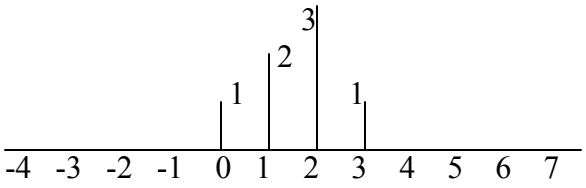
- Reflection of $h(k)$ resulting in $h(-k)$
- Shifting of $h(-k)$ resulting in $h(n-k)$
- Element wise multiplication of the sequences $x(k)$ and $h(n-k)$
- Summation of the product sequence $x(k)\, h(n-k)$ resulting in the convolution value for $y(n)$

Example:

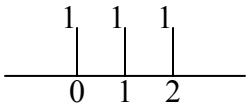
x(n) = {1, 2, 3, 1}
h(n) = {1, 1, 1}

length(y(n)) = length(x(n)) + length(y(n)) – 1
= 4 + 3 – 1 = 6

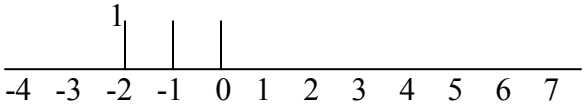
x(k)



h(k)

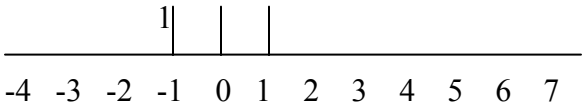


n=0; h(-k)



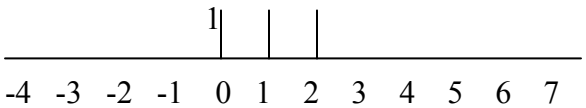
$$y(0) = \sum_{k=-\infty}^{\infty} x(k)\, h(-k) = 1$$

n=1; h(1-k)



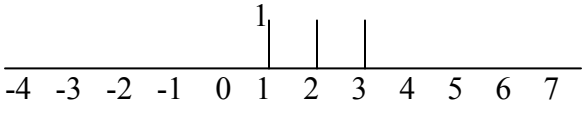
$$y(1) = \sum_{k=-\infty}^{\infty} x(k)\, h(1-k) = 1+2=3$$

n=2; h(2-k)



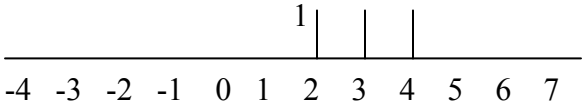
$$y(2) = \sum_{k=-\infty}^{\infty} x(k)\, h(2-k) = 1+2+3=6$$

n=3; h(3-k)



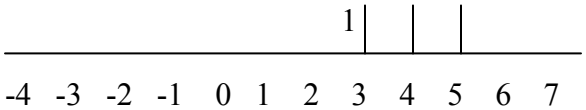
$$y(3) = \sum_{k=-\infty}^{\infty} x(k)\, h(3-k) = 2+3+1=6$$

n=4; h(4-k)

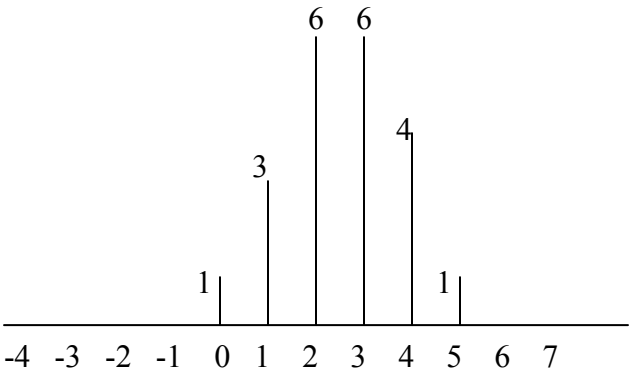


$$y(4) = \sum_{k=-\infty}^{\infty} x(k)\, h(4-k) = 3+1=4$$

n=5; h(5-k)



$$y(5) = \sum_{k=-\infty}^{\infty} x(k)\, h(5-k) = 1$$



$y(n) = \{1, 3, 6, 6, 4, 1\}$

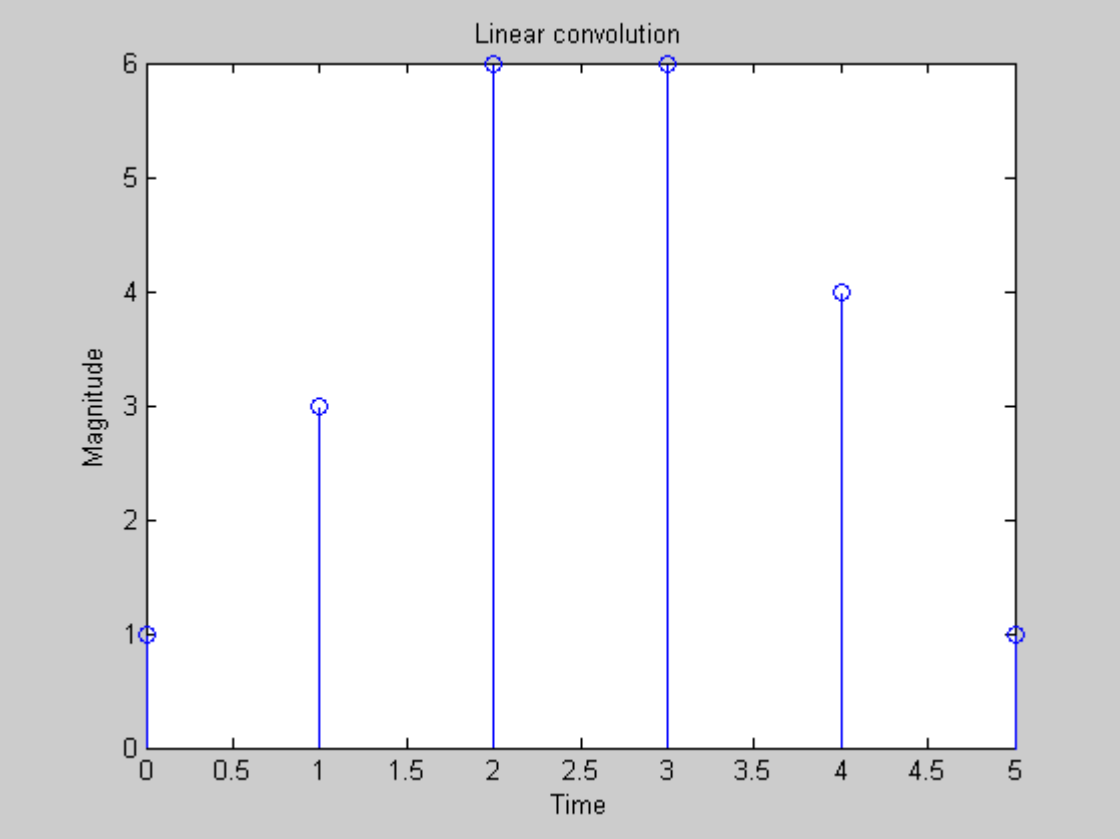
Program:

```
clc;                                    % Clear screen
x1 = input('Enter the 1st seq:');       % Get the first sequence
x2 = input('Enter the 2nd seq:');       % Get the second sequence
y = conv(x1, x2);                       % Convolve two signals
disp('The linear convolution of two sequences:'); % Displays the result
disp(y);                                % Displays the result on command window
n = 0:length(y)-1;                      % Defines the length for x-axis
stem(n, y);                             % Plots the output
xlabel('Time');                         % Name the x-axis as Time
ylabel('Magnitude');                    % Name the y-axis as Magnitude
title('Linear convolution');            % Title as "Linear convolution"
```

Output:

```
Enter the 1st seq:[1 2 3 1]
Enter the 2nd seq:[1 1 1]
The linear convolution of two sequences:
    1     3     6     6     4     1
```

Graph:


Linear convolution
Magnitude
Time
0
1
2
3
4
5
6
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

EXPERIMENT NO-4

AIM: TO IMPLEMENT CIRCULAR CONVOLUTION OF TWO GIVEN SEQUENCES

Circular convolution:

Let $x_1(n)$ and $x_2(n)$ are finite duration sequences both of length N with DFT's $X_1(k)$ and $X_2(k)$. Convolution of two given sequences $x_1(n)$ and $x_2(n)$ is given by the equation,

$$x_3(n) = IDFT[X_3(k)]$$

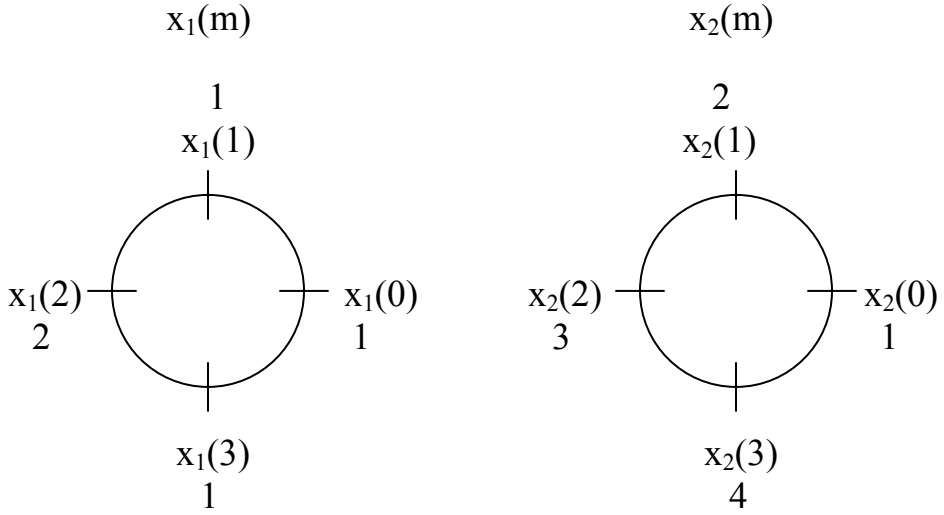
$$X_3(k) = X_1(k)\, X_2(k)$$

$$x_3(n) = \sum_{m=0}^{N-1} x_1(m)\, x_2((n-m))_N$$

Example:

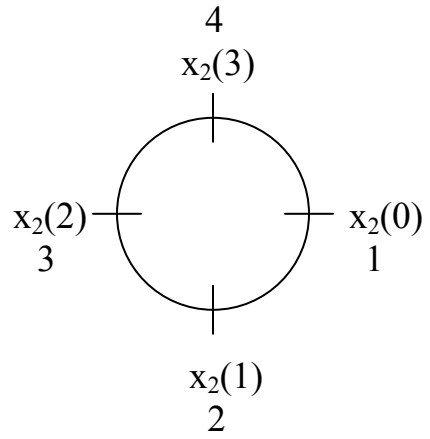
Let's take $x_1(n) = \{1, 1, 2, 1\}$ and
$x_2(n) = \{1, 2, 3, 4\}$

Arrange $x_1(n)$ and $x_2(n)$ in circular fashion as shown below.



To get $x_2(-m)$, rotate $x_2(m)$ by 4 samples in clockwise direction.

$x_2(-m)$

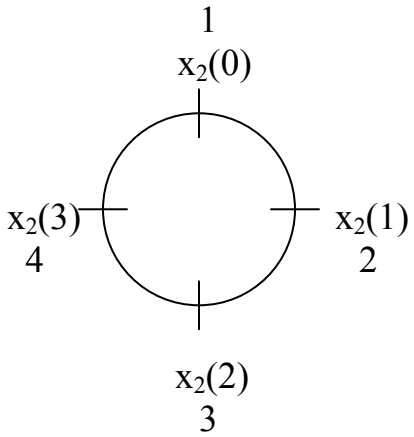


$$x_3(0) = x_1(m)\, x_2(-m)$$
$$= x_1(0)\, x_2(0) + x_1(1)\, x_2(3) + x_1(2)\, x_2(2) + x_1(3)\, x_2(1)$$
$$= 1 + 4 + 6 + 2$$
$$x_3(0) = 13$$

Keep $x_1(m)$ constant and rotate $x_2(-m)$ once to compute further values.

To get $x_3(1)$ rotate $x_2(-m)$ by one sample in anti-clockwise direction

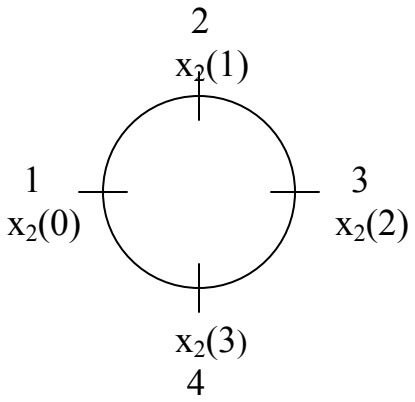
$x_2(1-m)$



$$x_3(1) = x_1(m)\, x_2(1-m)$$
$$= x_1(0)\, x_2(1) + x_1(1)\, x_2(0) + x_1(2)\, x_2(3) + x_1(3)\, x_2(2)$$
$$= 2 + 1 + 8 + 3$$
$$x_3(1) = 14$$

To get $x_3(2)$ rotate $x_2(1\text{-}m)$ by one sample in anti-clockwise direction

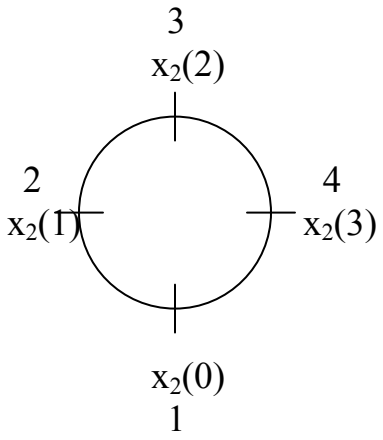
$x_2(2\text{-}m)$



$x_3(2) = x_1(m)\ x_2(2\text{-}m)$
$= x_1(0)\ x_2(2) + x_1(1)\ x_2(1) + x_1(2)\ x_2(0) + x_1(3)\ x_2(3)$
$= 3 + 2 + 2 + 4$
$x_3(2) = 11$

To get $x_3(3)$ rotate $x_2(2\text{-}m)$ by one sample in anti-clockwise direction

$x_2(3\text{-}m)$



$x_3(3) = x_1(m)\ x_2(3\text{-}m)$
$= x_1(0)\ x_2(3) + x_1(1)\ x_2(2) + x_1(2)\ x_2(1) + x_1(3)\ x_2(0)$
$= 4 + 3 + 4 + 1$
$x_3(3) = 12$

The convoluted signal is,
$x_3(n) = \{13, 14, 11, 12\}$

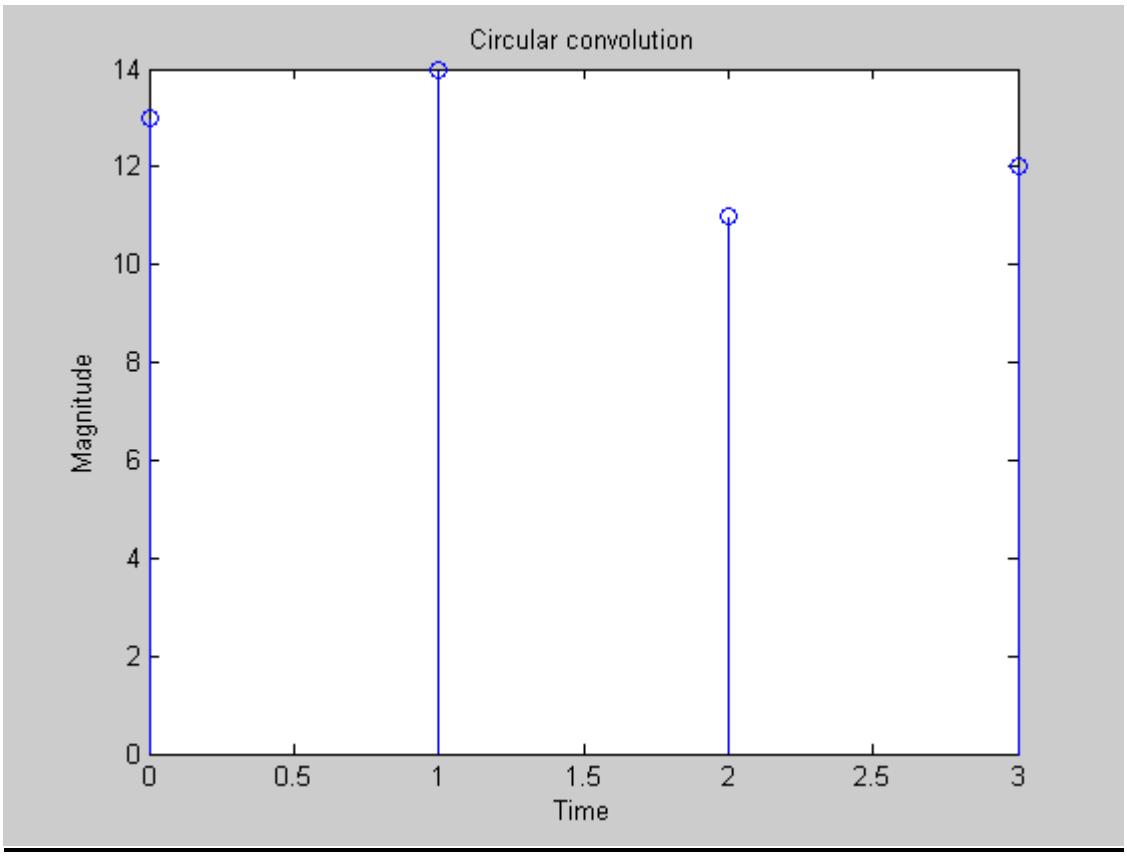
Program:

```
clc;                                    % Clear screen
x1 = input('Enter the first sequence:');    % Get the first sequence
x2 = input('Enter the second sequence:');   % Get the second sequence
N1 = length(x1);                        % Returns the length of first sequence
N2 = length(x2);                        % Returns the length of second sequence
N = max(N1,N2);                         % Get the Maximum length out of two signals
N3 = N1-N2;
% Get equal length sequence
if(N3>0)
x1 = [x1,zeros(1, N3)];                 % Pad zeros to the first sequence if N2>N1
else
x2 = [x2,zeros(1, -N3)];                % Pad zeros to the second sequence is N1>N2
end
% Computation of circular convolution
for m=1:N
   y(m) = 0;
      for n=1:N
         i=m-n+1;
if(i<=0)
    i=N+i;
end
y(m) = y(m)+x1(n)*h(i);
      end
end
disp('The circular convolution of two given sequences is:');
disp(y);                                % Displays the result on command window
n=0:N-1;                                % Get the length for x-axis
stem(n, y);                             % Plot the result
xlabel('Time');                         % Name the x-axis as “Time”
ylabel('Magnitude');                    % Name the y-axis as “Magnitude”
title('Circular convolution');          % Title as Circular convolution
```

Output:

```
Enter the first sequence: [1 1 2 1]
Enter the second sequence: [1 2 3 4]
The circular convolution of two given sequences is:
   13    14    11    12
```

Graph:


Circular convolution
Magnitude
Time
0
2
4
6
8
10
12
14
0
0.5
1
1.5
2
2.5
3

EXPERIMENT NO-5

AIM: AUTOCORRELATION OF A GIVEN SEQUENCE AND VERIFICATION OF ITS PROPERTIES

Correlation: Correlation determines the degree of similarity between two signals. If the signals are identical, then the correlation coefficient is 1; if they are totally different, the correlation coefficient is 0, and if they are identical except that the phase is shifted by exactly 180^0(i.e. mirrored), then the correlation coefficient is -1.

Autocorrelation: The Autocorrelation of a sequence is correlation of a sequence with itself. The autocorrelation of a sequence $x(n)$ is defined by,

$$R_{xx}(k) = \sum_{n=-\infty}^{\infty} x(n)\, x(n-k) \qquad k = 0, \pm1, \pm2, \pm3 \ldots$$

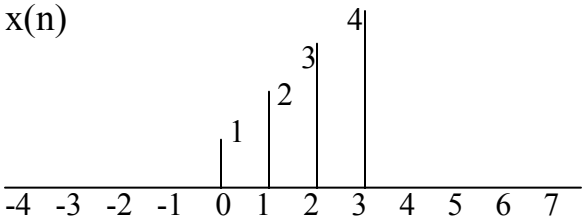
Where k is the shift parameter.

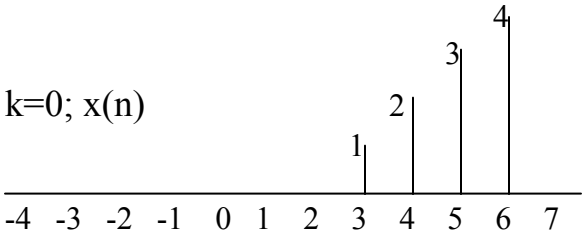
Or equivalently

$$R_{xx}(k) = \sum_{n=-\infty}^{\infty} x(n+k)\, x(n) \qquad k = 0, \pm1, \pm2, \pm3 \ldots$$

Example:

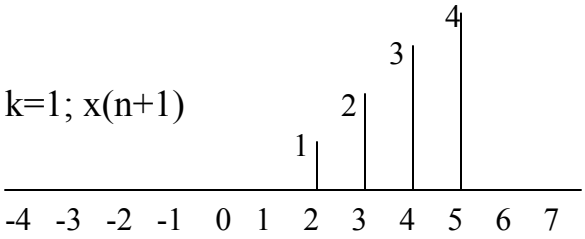
Let's take a sequence $x(n) = \{1, 2, 3, 4\}$





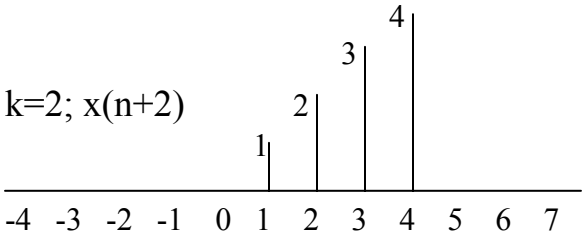
$$R_{xx}(0) = \sum_{n=-\infty}^{\infty} x(n)\, x(n) = 4$$

k=1; x(n+1)



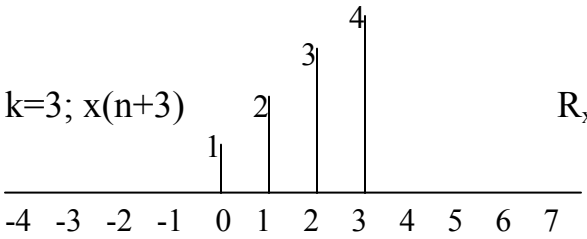
$$R_{xx}(1) = \sum_{n=-\infty}^{\infty} x(n)\, x(n+1) = 8+3=11$$

k=2; x(n+2)



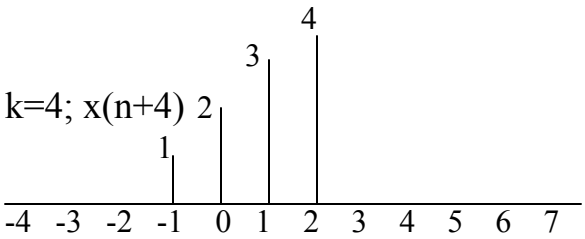
$$R_{xx}(2) = \sum_{n=-\infty}^{\infty} x(n)\, x(n+2) = 12+6+2=20$$

k=3; x(n+3)



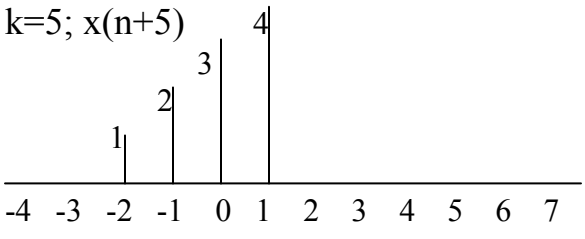
$$R_{xx}(3) = \sum_{n=-\infty}^{\infty} x(n)\, x(n+3) = 16+9+4+1=30$$

k=4; x(n+4)



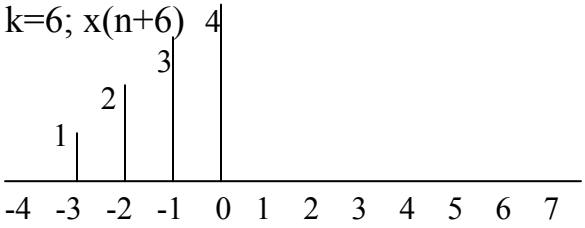
$$R_{xx}(4) = \sum_{n=-\infty}^{\infty} x(n)\, x(n+4) = 12+6+2=20$$

k=5; x(n+5)

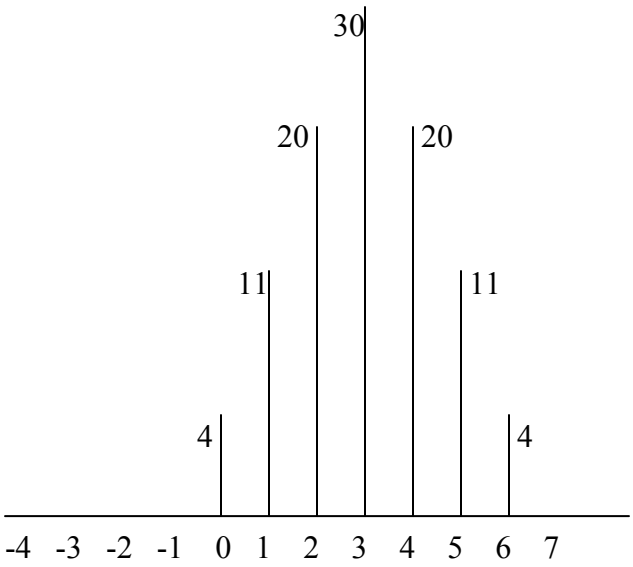


$$R_{xx}(5) = \sum_{n=-\infty}^{\infty} x(n)\, x(n+5) = 8+3=11$$

k=6; x(n+6)



$$R_{xx}(6) = \sum_{n=-\infty}^{\infty} x(n)\, x(n+6) = 4$$



$R_{xx}(k) = \{4, 11, 20, 30, 20, 11, 4\}$

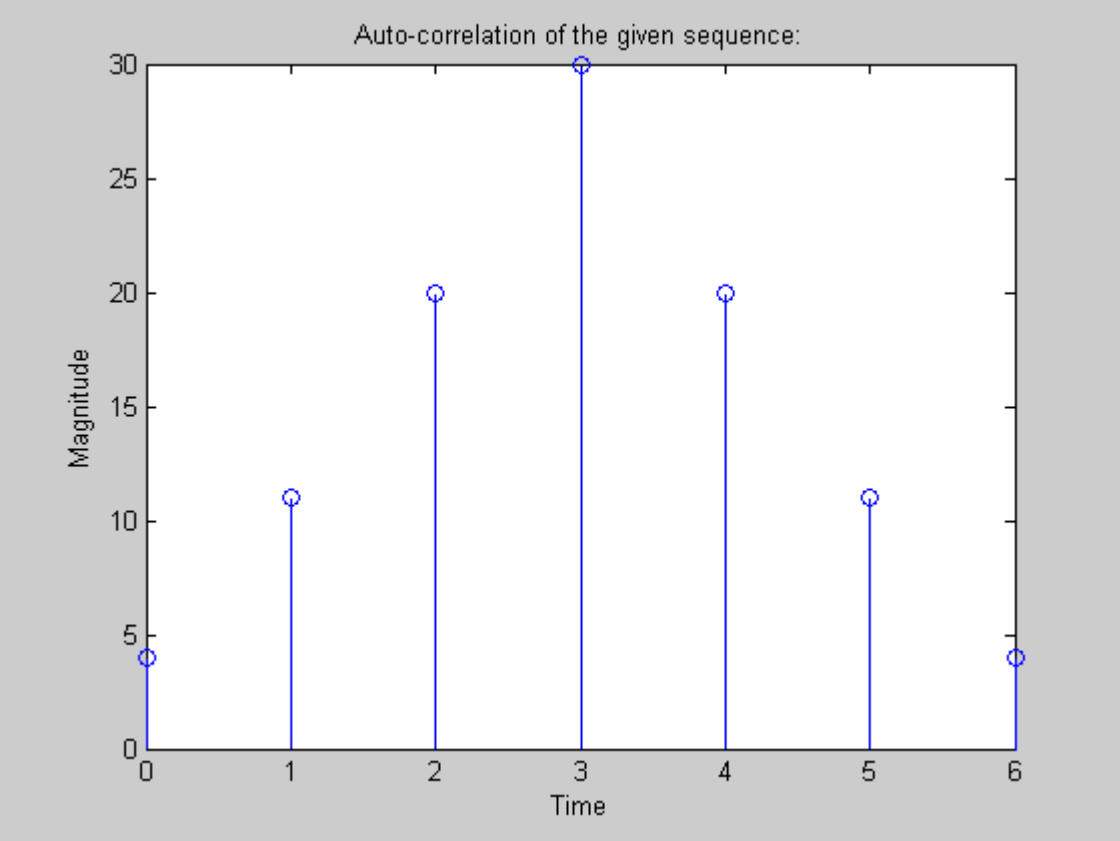
Program:

```
clc;                                                    % Clear screen
x = input('Enter the input sequence:');                 % Get the input sequence
Rxx = xcorr(x);                                         % Returns auto-correlated sequence
disp('Auto correlated sequence, Rxx:');                 % Display the result
disp(Rxx);                                              % Display the result on command window
t = 0:length(Rxx)-1;                                    % Length to plot the graph
stem(t,Rxx);                                            % plots the result on figure window
xlabel('Time');                                         % xlabel
ylabel('Magnitude');                                    % ylabel
title('Auto-correlation of the given sequence:'); % Title
```

Output:

```
Enter the input sequence: [1 2 3 4]
Auto correlated sequence, Rxx:
   4.0000   11.0000   20.0000   30.0000   20.0000   11.0000    4.0000
```

Graph:



Properties of Autocorrelation:

1. **Periodicity:** $R_{xx}(k_0) = R_{xx}(0)$ then $R_{xx}(k)$ is periodic with period k_0
2. Autocorrelation function is **symmetric.** i.e. $R_{xx}(m) = R_{xx}(-m)$
3. **Mean square value:** autocorrelation function at $k=0$, is equal to mean square value of the process. $R_{xx}(0) = E\{|x(n)|^2\} \geq 0$

EXPERIMENT NO-6

AIM: CROSS-CORRELATION OF A GIVEN SEQUENCE AND VERIFICATION OF ITS PROPERTIES

Correlation: Correlation determines the degree of similarity between two signals. If the signals are identical, then the correlation coefficient is 1; if they are totally different, the correlation coefficient is 0, and if they are identical except that the phase is shifted by exactly 180^0(i.e. mirrored), then the correlation coefficient is -1.

Cross-correlation: When two independent signals are compared, the procedure is known as *cross-correlation*. It is given by,

$$R_{xy}(k) = \sum_{n=-\infty}^{\infty} x(n)\, y(n-k) \qquad k = 0, \pm1, \pm2, \pm3 \ \ldots$$

Where k is the shift parameter.

Or equivalently

$$R_{yx}(k) = \sum_{n=-\infty}^{\infty} y(n+k)\, x(n)$$

Comparing above two equations, we find that,

$$R_{xy}(k) = R_{yx}(-k)$$

Where $R_{yx}(-k)$ is the folded version of $R_{xy}(k)$ about $k = 0$.

So, we can write Cross correlation of the two sequences is given by,

$$R_{xy}(k) = \sum_{n=-\infty}^{\infty} x(n)\, y[-(k-n)]$$

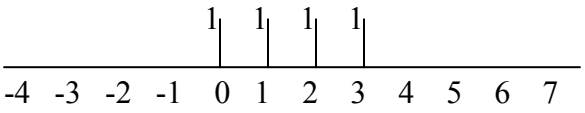
$$R_{xy}(k) = x(k) * y(-k) \qquad \ldots (1)$$

Equation (1) shows that cross correlation is the essentially the convolution of two sequences in which one of the sequences has been reversed.

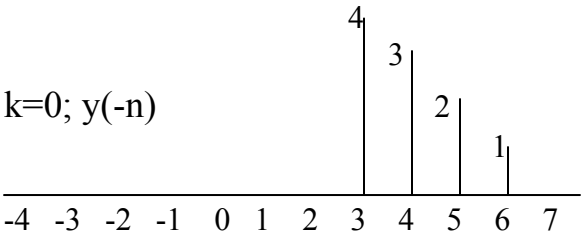
Example:

Let's take x(n) = {1, 1, 1, 1} and y(n) = {1, 2, 3, 4}

x(n)

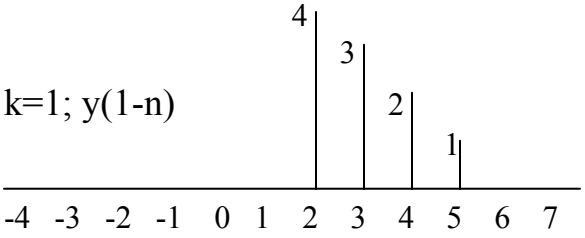


k=0; y(-n)



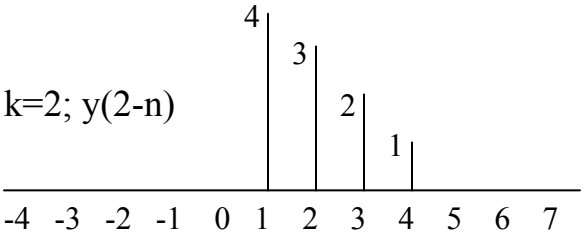
$$R_{xy}(0) = \sum_{n=-\infty}^{\infty} x(n)\, y(-n) = 4$$

k=1; y(1-n)



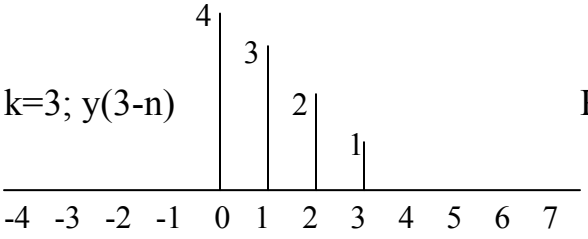
$$R_{xy}(1) = \sum_{n=-\infty}^{\infty} x(n)\, y(1-n) = 4+3=7$$

k=2; y(2-n)



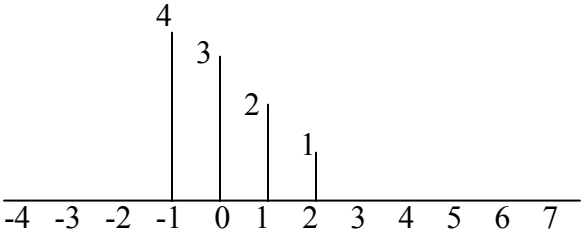
$$R_{xy}(2) = \sum_{n=-\infty}^{\infty} x(n)\, y(2-n) = 4+3+2=9$$

k=3; y(3-n)



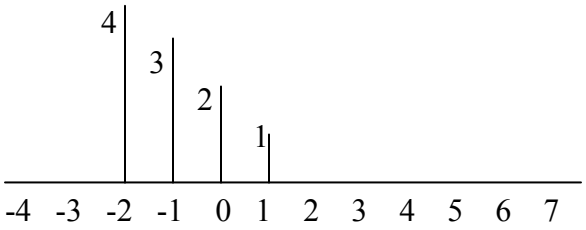
$$R_{xy}(3) = \sum_{n=-\infty}^{\infty} x(n)\, y(3-n) = 4+3+2+1=10$$

k=4; y(4-n)



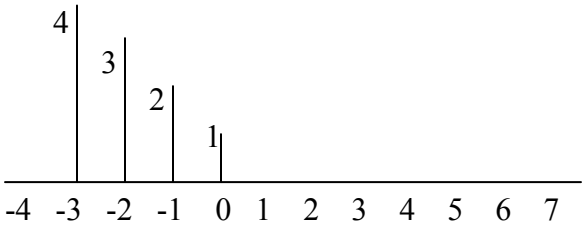
$$R_{xy}(4) = \sum_{n=-\infty}^{\infty} x(n)\, y(4-n) = 3+2+1=6$$

k=5; y(5-n)

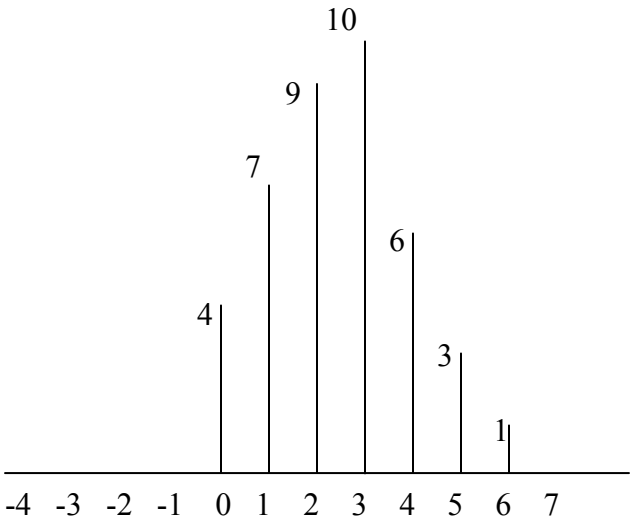


$$R_{xy}(5) = \sum_{n=-\infty}^{\infty} x(n)\, y(5-n) = 2+1 = 3$$

k=6; y(6-n)



$$R_{xy}(6) = \sum_{n=-\infty}^{\infty} x(n)\, y(6-n) = 1$$



$$R_{xy}(k) = \{4, 7, 9, 10, 6, 3, 1\}$$

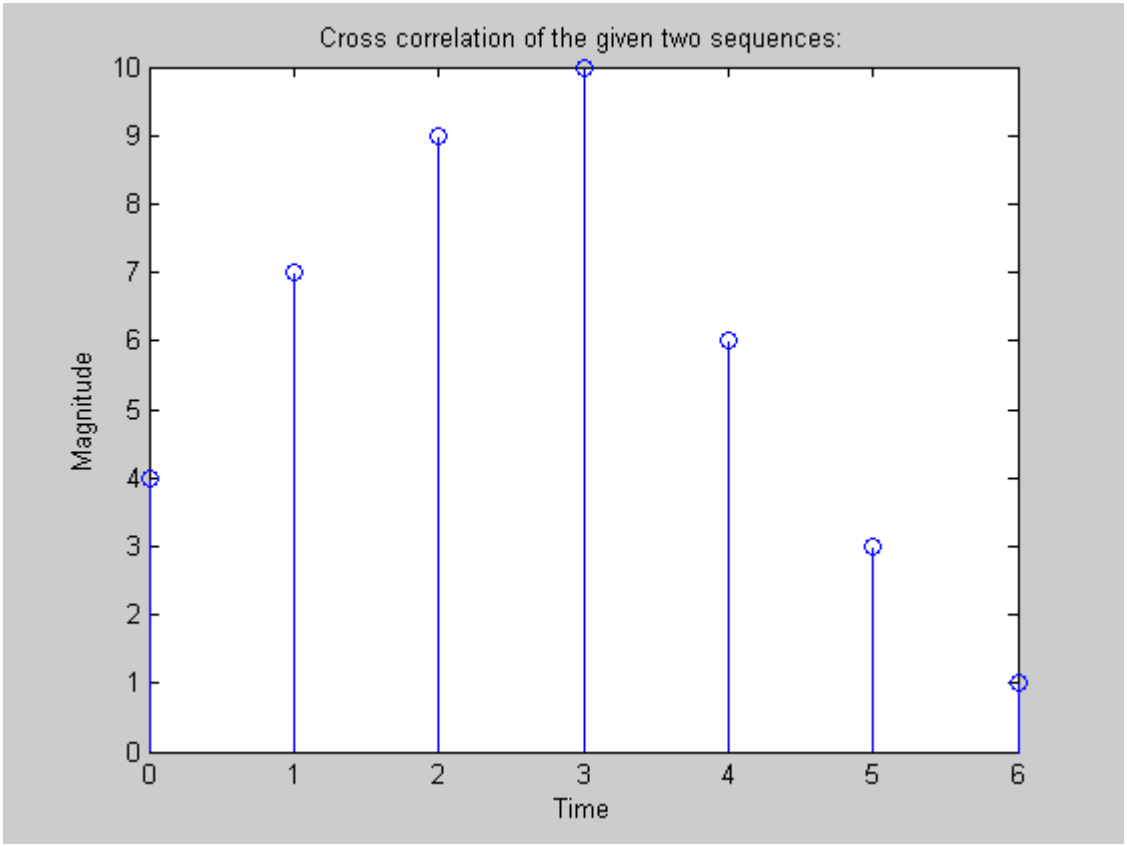
Program:

```
clc;                                    % Clear screen
x = input('Enter the first sequence:'); % Get the first sequence
y = input('Enter the second sequence:'); % Get the second sequence
Rxy = xcorr(x,y);                       % Returns cross correlated sequence
disp('Cross correlated sequence, Rxy:'); % Display the result
disp(Rxy);                              % Display the result on command window
t = 0:length(Rxy)-1;                    % Length to plot the graph
stem(t, Rxy);                           % plots the result on figure window
xlabel('Time');                         % xlabel
ylabel('Magnitude');                    % ylabel
title('Cross correlation of the given two sequences:'); % Title
```

Output:

```
Enter the first sequence: [1 1 1 1]
Enter the second sequence: [1 2 3 4]
Cross correlated sequence, Rxy:
   4.0000    7.0000    9.0000   10.0000    6.0000    3.0000    1.0000
```

Graph:



Properties of cross correlation:

1. $R_{xy}(k)$ is always a real valued function which may be a positive or negative.
2. $R_{xy}(-k) = R_{yx}(k)$
3. $|R_{xy}(k)|^2 \leq R_{xx}(0)\, R_{yy}(0)$
4. $|R_{xy}(-k)| \leq [1/2]\, [R_{xy}(-k) + R_{xy}(-k)]$
5. When $R_{xy}(k) = 0$, $x(n)$ and $y(n)$ are said to be uncorrelated or they said to be statistically independent.
6. $R_{xy}(k)$ may not be necessarily have a maximum at $k=0$ nor $R_{xy}(k)$ an even function.

EXPERIMENT NO-7

AIM: TO SOLVE A GIVEN DIFFERENCE EQUATION

A General N^{th} order Difference equations looks like,

$$y[n] + a_1y[n-1] + a_2y[n-2] + \ldots + a_Ny[n-N) = b_0x[n] + b_1x[n-1] + \ldots. + b_Mx[n-M]$$

Here $y[n]$ is "Most advanced" output sample and $y[n-m]$ is "Least advanced" ouput sample.
The difference between these two index values is the order of the difference equation.
Here we have: $n-(n-N) = N$

We can rewrite the above equation as,

$$y[n] + \sum a_i\, y[n-i] = \sum b_i\, x[n-i]$$

$y[n]$ becomes,

$$y[n] = -\sum a_i\, y[n-i] + \sum b_i\, x[n-i]$$

Example:

$$y[n+2] - 1.5y[n+1] + y[n] = 2x[n]$$

In general we start with the "Most advanced" output sample. Here it is $y[n+2]$.
So, here we need to subtract 2 from each sample argument. We get

$$y[n] - 1.5y[n-1] + y[n-2] = 2x[n-2]$$

$$\Rightarrow \quad y[n] = 1.5y[n-1] - y[n-2] + 2x[n-2]$$

Lets assume our input $x[n] = u[n] = \begin{cases} 0 & x<0 \\ 1 & x\geq 0 \end{cases}$

In our example we have taken $x[n] = u[n] = \begin{cases} 0 & x<0 \\ 1 & 0\leq x<10 \end{cases}$

We need N past values to solve N^{th} order difference equation.
$y[-2] = 1$
$y[-1] = 2$

Compute y[n] for various values of n

y[0] = 1.5y[-1] - y[-2] + 2x[-2]
= 1.5*2 – 1 + 2*0
y[0] = 2

y[1] = 1.5y[0] - y[-1] + 2x[-1]
= 1.5*2 - 2 + 2*0
y[1] = 1

y[2] = 1.5y[1] - y[0] + 2x[0]
= 1.5*1 – 2 + 2*1
y[2] = 1.5

And so on…

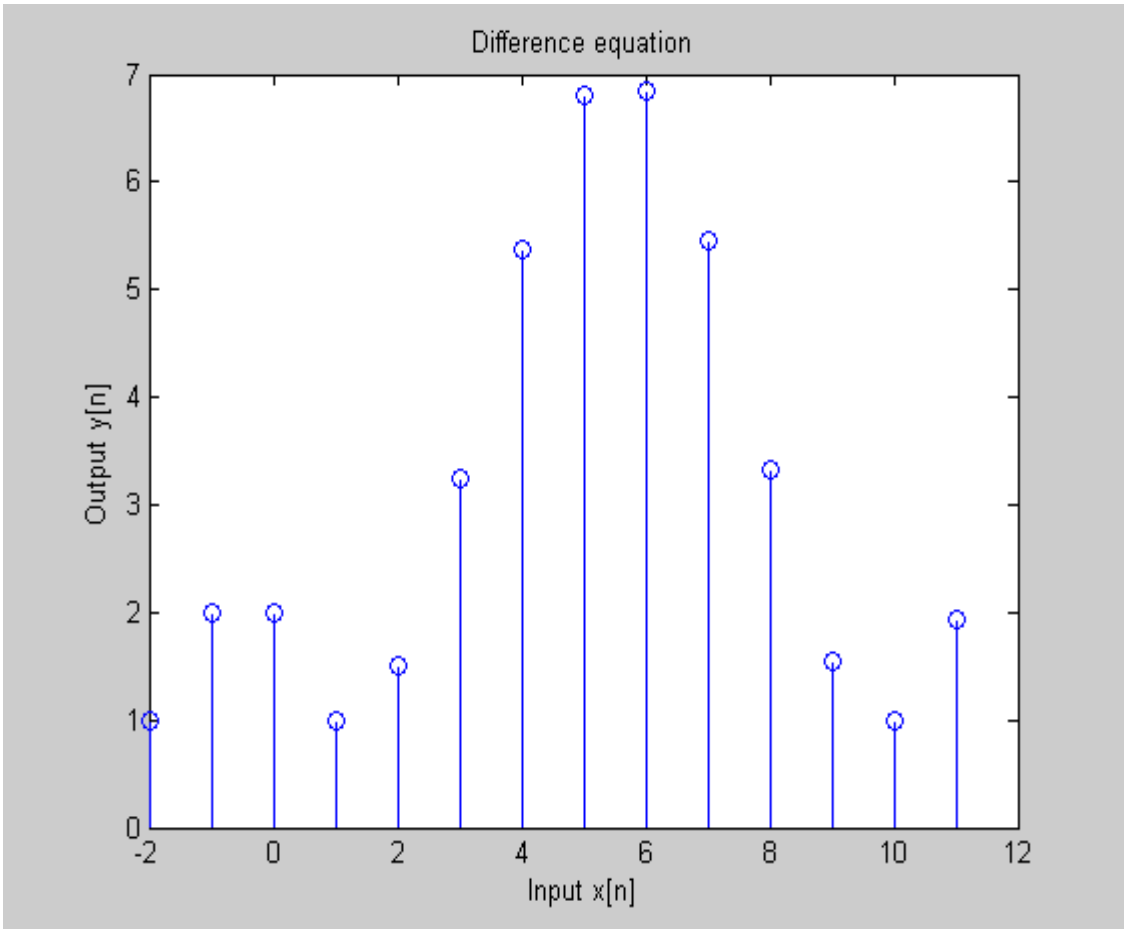
Program:

```
clc;                          % Clear screen
x = [0 0 ones(1, 10)];        % Input x[n] = u[n]
y_past = [1 2];               % Past values to solve difference equation
y(1) = y_past(1);             % y[1] <= y[-2]
y(2) = y_past(2);             % y[2] <= y[-1]

for k = 3:(length(x)+2)       % Compute y[n] for various values of n
   y(k) = 1.5*y(k-1) - y(k-2) + 2*x(k-2);
end                           % End of for loop

disp('Solution for the given difference equation:');
disp(y);                      % Display the result on command window
subplot(1,1,1);               % Divide the window to plot the result
stem (-2:(length(y)-3),y);    % Plot the result
xlabel('Input x[n]');         % Name x-axis as Input x[n]
ylabel('Output y[n]');        % Name y-axis as Output y[n]
title('Difference equation'); % Title as "Difference equation"
```

Output:

```
Solution for the given difference equation:
  Columns 1 through 7
    1.0000    2.0000    2.0000    1.0000    1.5000    3.2500    5.3750

  Columns 8 through 14
    6.8125    6.8438    5.4531    3.3359    1.5508    0.9902    1.9346
```

Graph:


Difference equation
Output y[n]
Input x[n]
0
1
2
3
4
5
6
7
-2
0
2
4
6
8
10
12

EXPERIMENT NO-8

AIM: TO COMPUTE N-POINT DFT OF A GIVEN SEQUENCE AND TO PLOT MAGNITUDE AND PHASE SPECTRUM.

Discrete Fourier Transform: The Discrete Fourier Transform is a powerful computation tool which allows us to evaluate the Fourier Transform $X(e^{j\omega})$ on a digital computer or specially designed digital hardware. Since $X(e^{j\omega})$ is continuous and periodic, the DFT is obtained by sampling one period of the Fourier Transform at a finite number of frequency points. Apart from determining the frequency content of a signal, DFT is used to perform linear filtering operations in the frequency domain.

The sequence of N complex numbers $x_0,..., x_{N-1}$ is transformed into the sequence of N complex numbers $X_0, ..., X_{N-1}$ by the DFT according to the formula:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \qquad k = 0,1, \ldots. N-1$$

Example:

Lets assume the input sequence x[n] = [1 1 0 0]

We have,

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \qquad k = 0,1, \ldots. N-1$$

For k = 0,

$$X(0) = \sum_{n=0}^{3} x(n) = x(0) + x(1) + x(2) + x(3)$$

$$X(0) = 1+1+0+0 = 2$$

For k = 1,

$$X(1) = \sum_{n=0}^{3} x(n)e^{-j\pi n/2} = x(0) + x(1)\, e^{-j\pi/2} + x(2)\, e^{-j\pi} + x(3)\, e^{-j3\pi/2}$$

$$= 1 + \cos(\pi/2) - j\sin(\pi/2)$$

$$X(1) = 1 - j$$

For k = 2,

$$X(2) = \sum_{n=0}^{3} x(n)e^{-j\pi n} = x(0) + x(1)\, e^{-j\pi} + x(2)\, e^{-j2\pi} + x(3)\, e^{-j3\pi}$$

$$= 1 + \cos\pi - j\sin\pi$$

$$X(2) = 1-1 = 0$$

For k = 3,

$$X(3) = \sum_{n=0}^{3} x(n)e^{-j3n\pi/2} = x(0) + x(1)\, e^{-j3\pi/2} + x(2)\, e^{-j3\pi} + x(3)\, e^{-j9\pi/2}$$

$$= 1 + \cos(3\pi/2) - j\sin(3\pi/2)$$

$$X(3) = 1 + j$$

The DFT of the given sequence is,

$X(k) = \{ 2, 1-j, 0, 1+j \}$

To find Magnitude of X(k):

Magnitude= $\mathbf{(a^2+b^2)^{1/2}}$

Where a and b are real and imaginary parts respectively

To fine Phase of X(k):

Phase=$\mathbf{\tan^{-1}(b/a)}$

Program:

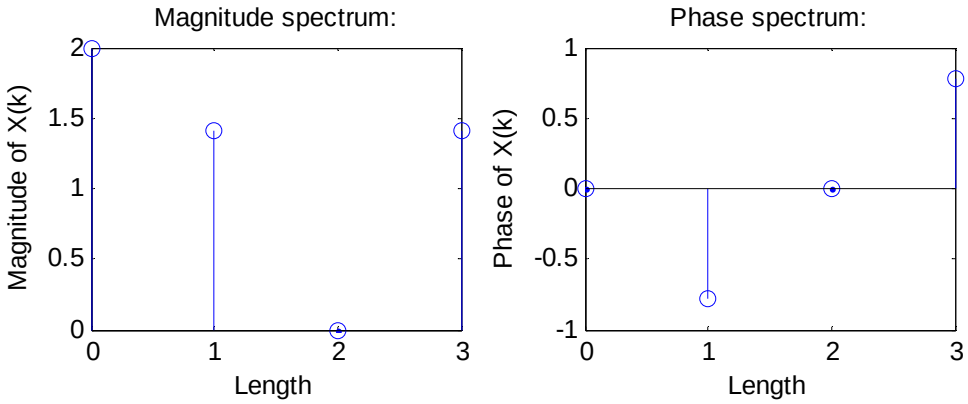
```
clc;                                        % Clear screen
x1 = input('Enter the sequence:');          % Get the input sequence
n = input('Enter the length:');             % Get the value of N
m = abs(fft(x1,n));                         % Computes the DFT using FFT algorithm
disp('N-point DFT of a given sequence:');   % Display the results
disp(m);                                    % Displays the result on command window
N = 0:1:n-1;                                % Decides the length to plot the results

subplot(2,2,1);                             % Divide the figure window to plot the
                                            % results
stem(N,m);                                  % Plots the magnitude spectrum
xlabel('Length');                           % Name x-axis as "Length"
ylabel('Magnitude of X(k)');                % Name y-axis as "Magnitude of X(k)"
title('Magnitude spectrum:');               % Title as "Magnitude spectrum"
an = angle(fft(x1,n));                      % Get the angle of the output sequence X(k)
disp(an);
subplot(2,2,2);                             % Divide the figure window to plot the
                                            % results
stem(N, an);                                % Plots the phase spectrum
```

```
xlabel('Length');                    % Name x-axis as “Length”
ylabel('Phase of X(k)');             % Name y-axis as “Phase of X(k)”
title('Phase spectrum:');            % Title as “Phase spectrum”
```

Output:

```
Enter the sequence:[1 1 0 0]
Enter the length:4
Magnitude of X(k)
   2.0000    1.4142         0    1.4142

Phase of X(k)
         0   -0.7854         0    0.7854
```

Graph:



EXPERIMENT NO-9

AIM: LINEAR CONVOLUTION OF TWO GIVEN SEQUENCES USING DFT AND IDFT

Theory: Convolution is an integral concatenation of two signals. It has many applications in numerous areas of signal processing. The most popular application is the determination of the output signal of a linear time-invariant system by convolving the input signal with the impulse response of the system.
Note that convolving two signals is equivalent to multiplying the Fourier Transform of the two signals.

Mathematic Formula:
The linear convolution of two continuous time signals $x(t)$ and $h(t)$ is defined by

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau)\, d\tau$$

For discrete time signals $x(n)$ and $h(n)$, is defined by

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$$

Where $x(n)$ is the input signal and **$h(n)$** is the impulse response of the system.

Example:
$x_1(n) = \{1, 1, 2\}$
$x_2(n) = \{1, 2\}$

For linear convolution,
Length N = Length(x_1) + Length(x_2) - 1
N = 3 + 2 – 1 = 4

Convolution of two sequences $x_1(n)$ and $x_2(n)$ is,
$x_3(n) = IDFT[X_3(k)]$
$x_3(n) = IDFT[X_1(k)\, X_2(k)]$

Where,
$X_1(k) = DFT\ [x_1(n)]$

$X_2(k) = DFT\ [x_2(n)]$

$$X_1(k) = \sum_{n=0}^{N-1} x_1(n)e^{-j2\pi kn/N} \qquad k= 0, 1, 2, \dots , N-1$$

Given $x_1(n) = \{1, 1, 2\}$ and N=4

$$X_1(0) = \sum_{n=0}^{3} x_1(n) = 1 + 1 + 2 = 4$$

$$X_1(1) = \sum_{n=0}^{5} x_1(n)e^{-j\pi n/2} = 1 - j - 2 = -1 - j$$

$$X_1(2) = \sum_{n=0}^{5} x_1(n)e^{-j\pi n} = 1 - 1 + 2 = 2$$

$$X_1(3) = \sum_{n=0}^{5} x_1(n)e^{-j3\pi n/2} = 1 + j - 2 = -1 + j$$

$X_1(k) = \{4, -1-j, 2, -1+j\}$

Now,

$$X_2(k) = \sum_{n=0}^{N-1} x_2(n)e^{-j2\pi kn/N} \qquad k= 0, 1, 2, \dots , N-1$$

Given $x_2(n) = \{1, 2\}$ and N=4

$$X_2(0) = \sum_{n=0}^{3} x_2(n) = 1 + 2 = 3$$

$$X_2(1) = \sum_{n=0}^{3} x_2(n)\, e^{-j\pi n/2} = 1 + 2(-j) = 1 - j2$$

$$X_2(2) = \sum_{n=0}^{3} x_2(n)\, e^{-j\pi n} = 1 + 2(-1) = -1$$

$$X_2(3) = \sum_{n=0}^{3} x_2(n)\, e^{-j3\pi n/2} = 1 + 2(j) = 1 + j2$$

$X_2(k) = \{3, 1-j2, -1, 1+j2\}$

We know that,
$X_3(k) = X_1(k)\, X_2(k)$
$X_3(k) = \{12, -3+j, -2, -3-j\}$

Convolution of two given sequences is,
$x_3(n) = IDFT[X_3(k)]$

$$x_3(n) = (1/N) \sum_{k=0}^{N-1} X_3(k) e^{j2\pi kn/N} \qquad n = 0, 1, 2, \ldots, N-1$$

$$x_3(0) = (1/4) \sum_{k=0}^{3} X_3(k) = (1/4)\,[12 - 3 + j - 2 - 3 - j] = 1$$

$$x_3(1) = (1/4) \sum_{k=0}^{3} X_3(k) e^{j\pi k/2}$$

$$x_3(1) = (1/4)\,[12 + (-3 + j)\, j + (-2)\,(-1) + (-3 - j)\,(-j)] = 3$$

$$x_3(2) = (1/4) \sum_{k=0}^{3} X_3(k) e^{j\pi k}$$

$$x_3(2) = (1/4)\,[12 + (-3 + j)\,(-1) + (-2)\,(1) + (-3 - j)\,(-1)] = 4$$

$$x_3(3) = (1/4) \sum_{k=0}^{3} X_3(k) e^{j3\pi k/2}$$

$$x_3(3) = (1/4)\,[12 + (-3 + j)\,(-j) + (-2)\,(-1) + (-3 - j)\,(j)] = 4$$

Convoluted sequence of two given sequences is,
$x_3(n) = \{1, 3, 4, 4\}$

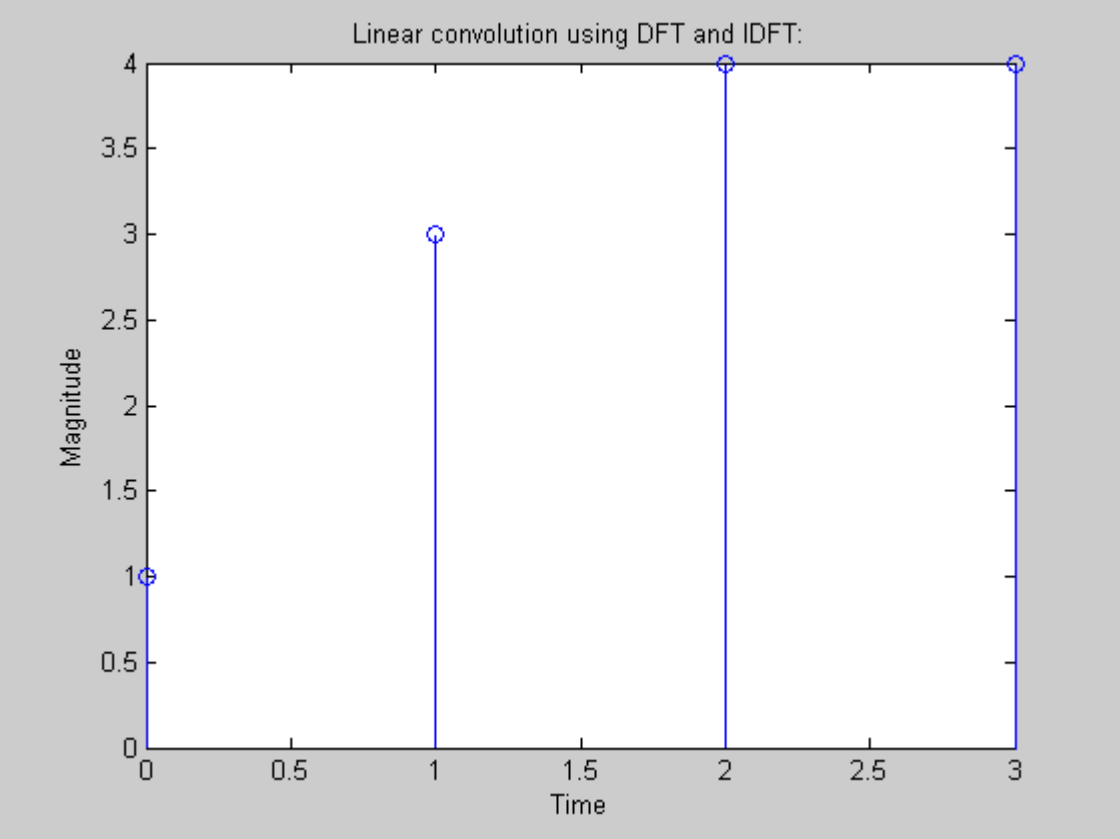
Program:

```
clc;                                    % Clear screen
x1 = input('Enter the 1st seq:');       % Get the first sequence
x2 = input('Enter the 2nd seq:');       % Get the second sequence
n = length(x1) + length(x2)-1;          % Get the length of the sequence
x1 = fft(x1,n);                         % Compute the DFT of  x1 using FFT algorithm
x2 = fft(x2,n);                         % Compute the DFT of  x2 using FFT algorithm
y = x1.*x2;                             % Multiply two DFT's
yc = ifft(y,n);                         % Compute IDFT using IFFT algorithm
disp('Linear convolution using DFT and IDFT:'); % Display Linear convolution
disp(yc);                               % Displays the result on command window
N = 0:1:n-1;                            % Defines the length of x-axis to plot the result
subplot(1,1,1);                         % Divide the window to plot the result
stem(N,yc);                             % Plots the result
xlabel('Time');                         % Name the x-axis as Time
ylabel('Magnitude');                    % Name the y-axis as Magnitude
title('Linear convolution using DFT and IDFT:'); % Title
```

Output:

```
Enter the 1st seq: [1 1 2]
Enter the 2nd seq: [1 2]
Linear convolution using DFT and IDFT:
    1     3     4     4
```

Graph:


Linear convolution using DFT and IDFT:
Magnitude
Time
0
0.5
1
1.5
2
2.5
3
3.5
4

EXPERIMENT NO-10

AIM: TO IMPLEMENT CIRCULAR CONVOLUTION OF TWO GIVEN SEQUENCES USING DFT AND IDFT

Circular convolution:

Let $x_1(n)$ and $x_2(n)$ are finite duration sequences both of length N with DFT's $X_1(k)$ and $X_2(k)$. Convolution of two given sequences $x_1(n)$ and $x_2(n)$ is given by,

$x_3(n) = IDFT[X_3(k)]$
$x_3(n) = IDFT[X_1(k)\ X_2(k)]$

Where,
$X_1(k) = DFT\ [x_1(n)]$
$X_2(k) = DFT\ [x_2(n)]$

Example:

$x_1(n) = \{1, 1, 2, 1\}$
$x_2(n) = \{1, 2, 3, 4\}$

$$X_1(k) = \sum_{n=0}^{N-1} x_1(n)e^{-j2\pi kn/N} \qquad k= 0, 1, 2, \dots , N-1$$

Given $x_1(n) = \{1, 1, 2, 1\}$ and N=4

$$X_1(0) = \sum_{n=0}^{3} x_1(n) = 1 + 1 + 2 + 1 = 5$$

$$X_1(1) = \sum_{n=0}^{3} x_1(n)e^{-j\pi n/2} = 1 - j - 2 + j = -1$$

$$X_1(2) = \sum_{n=0}^{3} x_1(n)e^{-j\pi n} = 1 - 1 + 2 - 1 = 1$$

$$X_1(3) = \sum_{n=0}^{3} x_1(n)e^{-j3\pi n/2} = 1 + j - 2 - j = -1$$

$X_1(k) = \{5, -1, 1, -1\}$

Now,

$$X_2(k) = \sum_{n=0}^{N-1} x_2(n)e^{-j2\pi kn/N} \qquad k= 0, 1, 2, \dots , N-1$$

Given $x_2(n) = \{1, 2, 3, 4\}$ and N=4

$$X_2(0) = \sum_{n=0}^{3} x_2(n) = 1 + 2 + 3 + 4 = 10$$

$$X_2(1) = \sum_{n=0}^{3} x_2(n)\, e^{-j\pi n/2} = 1 + 2(-j) + 3(-1) + 4(j) = -2 + j2$$

$$X_2(2) = \sum_{n=0}^{3} x_2(n)\, e^{-j\pi n} = 1 + 2(-1) + 3(1) + 4(-1) = -2$$

$$X_2(3) = \sum_{n=0}^{3} x_2(n)\, e^{-j3\pi n/2} = 1 + 2(j) + 3(-1) + 4(-j) = -2 - j2$$

$X_2(k) = \{10, -2+j2, -2, -2-j2\}$

We know that,
$X_3(k) = X_1(k)\, X_2(k)$
$X_3(k) = \{50, 2 - j2, -2, 2 + j2\}$

Convolution of two given sequences is,
$x_3(n) = IDFT[X_3(k)]$

$$x_3(n) = (1/N) \sum_{k=0}^{N-1} X_3(k)e^{j2\pi kn/N} \qquad n = 0, 1, 2, \dots , N-1$$

$$x_3(0) = (1/4) \sum_{k=0}^{3} X_3(k) = (1/4)\,[50 + 2 - j2 - 2 + 2 + j2] = 13$$

$$x_3(1) = (1/4) \sum_{k=0}^{3} X_3(k)e^{j\pi k/2}$$

$$x_3(1) = (1/4)\,[50 + (2 - j2)\,j + (-2)\,(-1) + (2 + j2)\,(-j)] = 14$$

$$x_3(2) = (1/4) \sum_{k=0}^{3} X_3(k)e^{j\pi k}$$

$$x_3(2) = (1/4)\,[50 + (2 - j2)\,(-1) + (-2)\,(1) + (2 + j2)\,(-1)] = 11$$

$$x_3(3) = (1/4) \sum_{k=0}^{3} X_3(k)e^{j3\pi k/2}$$

$$x_3(3) = (1/4)\,[50 + (2 - j2)\,(-j) + (-2)\,(-1) + (2 + j2)\,(j)] = 12$$

Convoluted sequence of two given sequences is,

$x_3(n) = \{13, 14, 11, 12\}$

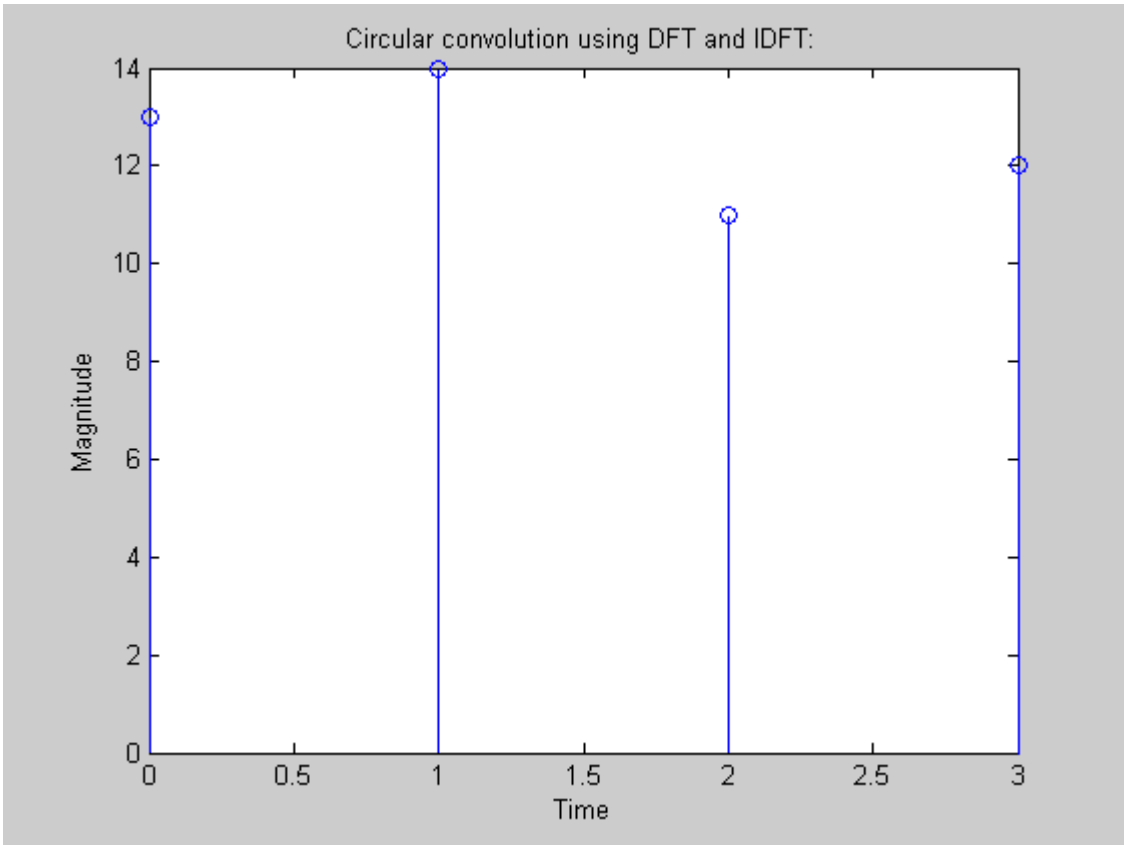
Program:

```
clc;                                    % Clear screen
x1 = input('Enter 1st sequence:');      % Get the first sequence
x2 = input('Enter 2nd sequence:');      % Get the second sequence
n = max(length(x1), length(x2));        % Get the maximum length of the two sequences
x1 = fft(x1,n);                         % Compute the DFT of the x1 using FFT algorithm
x2 = fft(x2,n);                         % Compute the DFT of the x2 using FFT algorithm
y = x1.*x2;                             % Multiply two DFT's
yc = ifft(y,n);                         % Compute the IDFT of mutliplied sequence to get
                                        % the convoluted sequence
disp('Circular convolution using DFT and IDFT:');   % Displays Circular convolution
disp(yc);                               % Displays the result on command window
N = 0:1:n-1;                            % Defines the length of x-axis to plot the result
subplot(1,1,1);                         % Divide the window to plot the result
stem(N,yc);                             % Plots the results
xlabel('Time');                         % Name the x-axis as "Time"
ylabel('Magnitude');                    % Name the y-axis as "Magnitude"
title('Circular convolution using DFT and IDFT:');   % Title
```

Output:

Enter 1st sequence:[1 1 2 1]
Enter 2nd sequence:[1 2 3 4]
Circular convolution using DFT and IDFT:
   13   14   11   12

Graph:



EXPERIMENT NO-11

AIM: DESIGN AND IMPLEMENTATION OF FIR FILTER TO MEET GIVEN SPECIFICATIONS (LOW PASS FILTER USING HAMMING WINDOW)

Finite Impulse Response (FIR) Filter: The FIR filters are of non-recursive type, whereby the present output sample is depending on the present input sample and previous input samples.
The transfer function of a FIR causal filter is given by,

$$H(z) = \sum_{n=0}^{N-1} h(n)z^{-n}$$

Where $h(n)$ is the impulse response of the filter.

The Fourier transform of $h(n)$ is

$$H(e^{jw}) = \sum_{n=0}^{N-1} h(n)e^{-jwn}$$

In the design of FIR filters most commonly used approach is using windows.
The desired frequency response $H_d(e^{jw})$ of a filter is periodic in frequency and can be expanded in Fourier series. The resultant series is given by,

$$h_d(n) = (1/2\pi) \int_{-\pi}^{\pi} H(e^{jw})e^{jwn}\, dw$$

And known as Fourier coefficients having infinite length. One possible way of obtaining FIR filter is to truncate the infinite Fourier series at $n = \pm [(N-1)/2]$
Where N is the length of the desired sequence.
The Fourier coefficients of the filter are modified by multiplying the infinite impulse response with a finite weighing sequence $w(n)$ called a window.

Where $w(n) = w(-n) \neq 0$ for $|n| \leq [(N-1)/2]$
$= 0$ for $|n| > [(N-1)/2]$

After multiplying $w(n)$ with $h_d(n)$, we get a finite duration sequence $h(n)$ that satisfies the desired magnitude response,

$h(n) = h_d(n)\, w(n)$ for $|n| \leq [(N-1)/2]$
$= 0$ for $|n| > [(N-1)/2]$

The frequency response $H(e^{jw})$ of the filter can be obtained by convolution of $H_d(e^{jw})$ and $W(e^{jw})$ is given by,

$$H(e^{jw}) = (1/2\pi) \int_{-\pi}^{\pi} H_d(e^{j\theta})\, W(e^{j(w-\theta)})\, d\theta$$

$$H(e^{jw}) = H_d(e^{jw}) * W(e^{jw})$$

Example:

Here we design a lowpass filter using hamming window.
Hamming window function is given by,

$$w_H(n) = 0.54 + 0.46 \cos((2\pi n)/(N-1)) \qquad \text{for } -(N-1)/2 \le n \le (N-1)/2$$

$$= 0 \qquad \text{otherwise}$$

The frequency response of Hamming window is,

$$W_H(e^{jw}) = 0.54[(\sin(wN/2))/(\sin(w/2))] + 0.23[\sin(wN/2 - \pi N/N - 1)/\sin(w/2 - \pi/N - 1)] + 0.23[\sin(wN/2 + \pi N/N - 1)/\sin(w/2 + \pi/N - 1)]$$

Program:

```
clc;                                % Clear screen
wp = input('Pass band edge freq:'); % Get Passband edge frequency
ws = input('Stop band edge freq:'); % Get Stopband edge frequency
tw = ws-wp;                         % Subtract PB frequency from SB frequency

N = ceil(6.6*pi/tw)+1;              % Compute length
wn = (hamming(N));                  % Returns N-point symmetric hamming window in
                                    % column vector
B = fir1(N-1,wp,wn);                % Designs (N-1)th order lowpass FIR filter and
                                    % returns filter coefficients in length N in vector B
disp('Impulse response coeff=');    % Displays Impulse response coefficients
disp(B);                            % Displays the coefficients on command window
[H,w] = freqz(B,1,256);             % Digital filter frequency response. This function
                                    % returns the N-point complex frequency response
                                    % vector H and the N-point frequency vector w in
                                    % radians/sample of the filter.
Mag = 20*log10(abs(H));             % Get the magnitude
plot(w/pi*pi, Mag);                 % Plot the Magnitude spectrum
xlabel('Frequency in radians in terms of pi');% Name x-axis
ylabel('Gain in db');               % Name y-axis as "Gain in db"
```

Output:

Pass band edge freq: 0.05*pi
Stop band edge freq: 0.4*pi
Impulse response coeff=
Columns 1 through 7

-0.0027 -0.0035 -0.0041 -0.0010 0.0105 0.0333 0.0666

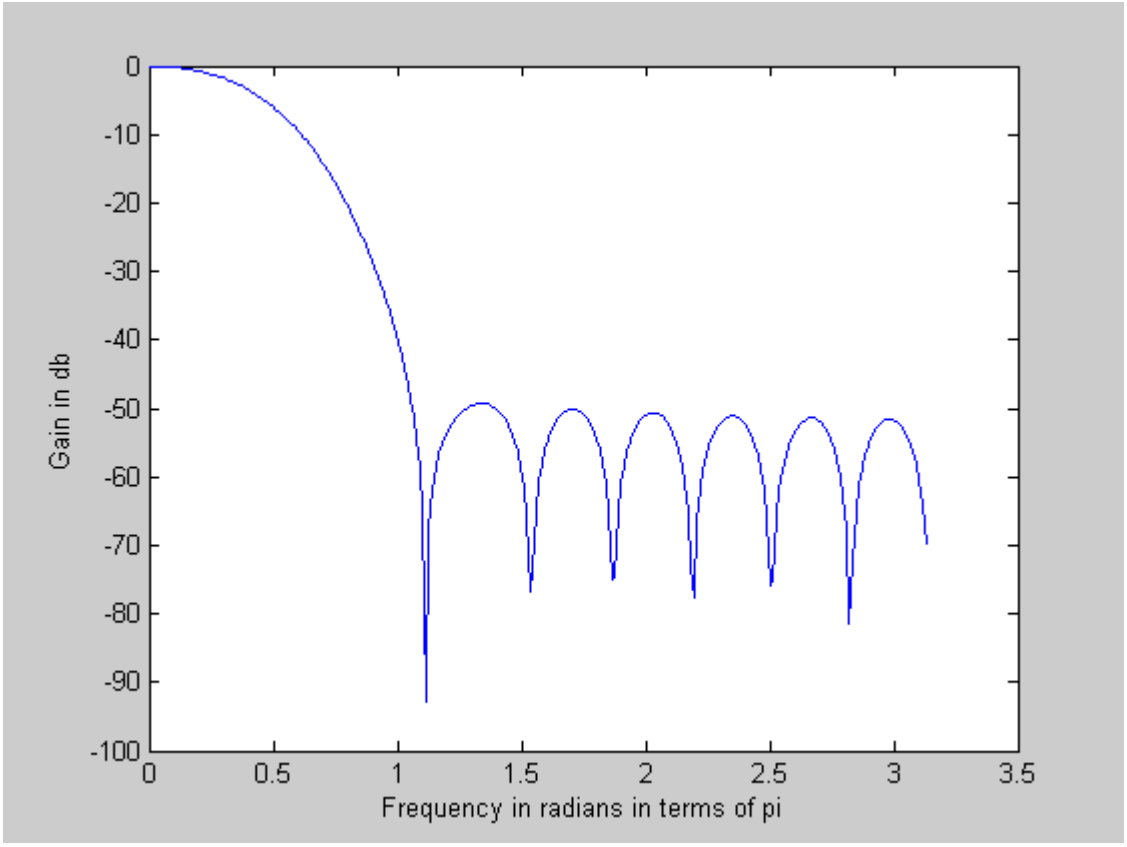
Columns 8 through 14

0.1046 0.1383 0.1580 0.1580 0.1383 0.1046 0.0666

Columns 15 through 20

0.0333 0.0105 -0.0010 -0.0041 -0.0035 -0.0027

Graph:



EXPERIMENT NO-12

AIM: DESIGN AND IMPLEMENTATION OF IIR FILTER TO MEET GIVEN SPECIFICATIONS

Basically digital filter is a linear time-invariant discrete time system.

Infinite Impulse Response(IIR) filter: IIR filters are of recursive type, whereby the present output sample depends on the present input, past input samples and output samples.

The impulse response $h(n)$ for a realizable filter is,

$$h(n) = 0 \qquad \text{for } n \leq 0$$

And for stability, it must satisfy the condition,

$$\sum_{n=0}^{\infty} | h(n) | < \infty$$

Example:

Let's design an analog Butterworth lowpass filter.

Steps to design an analog Butterworth lowpass filter.

1. From the given specifications find the order of the filter N.
2. Round off it to the next higher integer.
3. Find the transfer function $H(s)$ for $\Omega c = 1 \text{rad/sec}$ for the value of N.
4. calculate the value of cutoff frequency Ωc
5. find the transfer function $H_a(s)$ for the above value of Ωc by substituting $s \rightarrow (s/\Omega c)$ in $H(s)$.

Program:

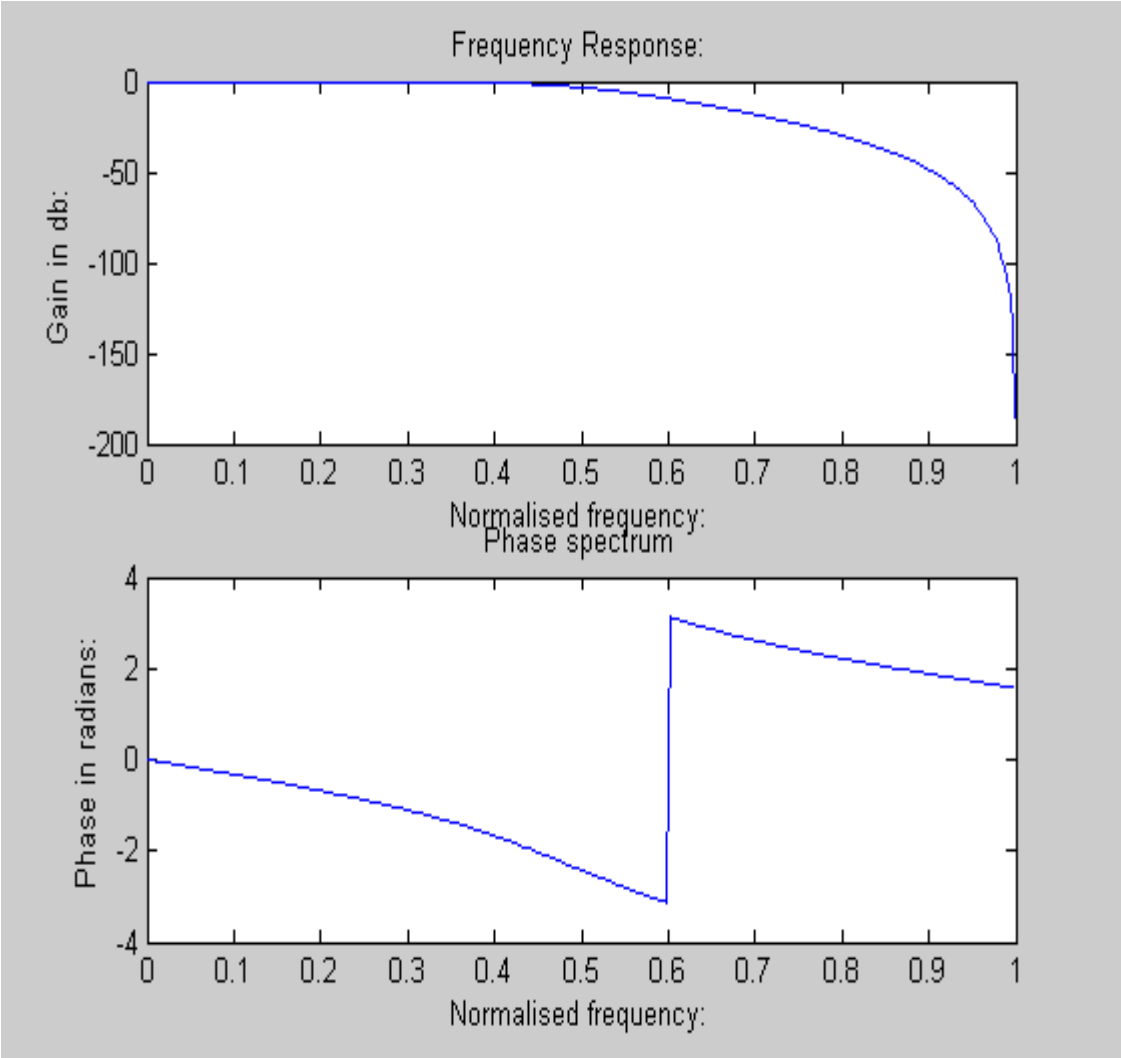
```
clc;                                          % Clear screen
Pa = input('Passband attenuation in DB:');    % Get the passband attenuation
Sa = input('Stopband attenuation in DB:');    % Get the stopband attenuation
Fpb = input('Passband edge frequency in Hz:'); % Get Passband edge frequency
Fsb = input('Stopband edge frequency in Hz:'); % Get Stopband edge frequency
fs = input('Sampling frequency:');            % Get Sampling frequency

wp = 2*Fpb/fs;                % Convert PB edge frequency in Hz to radians
ws = 2*Fsb/fs;                % Convert SB edge frequency in Hz to radians

[N,wn] = buttord(wp,ws,Pa,Sa); % Find cutoff frequency and order of the filter
disp('Order:');               % Display the Order
disp(N);                      % Display order N on command window
disp('Cutoff frequency:');    % Display Cutoff frequency
disp(wn);                     % Display Cutoff frequency on command window
```

```
[b,a] = butter(N,wn);                  % Get Numerator and denominator coefficients of
                                       % the filter
disp('b='); disp(b);                   % Display the numerator coefficients on command
                                       % window
disp('a='); disp(a);                   % Display the denominator coefficients on
                                       % command window
w = 0:0.01:pi;                         % Defines length for x-axis to plot the result
[h,om] = freqz(b,a,w,'whole');         % Frequency response of the filter
m = 20*log10(abs(h));                  % Absolute value of the frequency response vector
an = angle(h);                         % Get the phase of the frequency response vector

subplot(2,1,1);                        % Divide the figure window to plot the frequency
                                       % response
plot(om/pi,m);                         % Plot the response
xlabel('Normalised frequency:');       % Name x-axis as "Normalised frequency"
ylabel('Gain in db:');                 % Name y-axis as " Gain in db"
title('Frequency Response:');          % Title as Frequency Response
subplot(2,1,2);                        % Divide the figure window
plot(om/pi,an);                        % Plot the Phase spectrum
xlabel('Normalised frequency:');       % Name x-axis as "Normalised frequency"
ylabel('Phase in radians:');           % Name y-axis as Phase in radians
title('Phase spectrum');               % Title as Phase spectrum
```

Output:

```
Passband attenuation in DB:4
Stopband attenuation in DB:30
Passband edge frequency in Hz:400
Stopband edge frequency in Hz:800
Sampling frequency:2000
Order:
     3

Cutoff frequency:
    0.4914

b=
    0.1600    0.4800    0.4800    0.1600

a=
    1.0000   -0.0494    0.3340   -0.0045
```

Graph:


Frequency Response:
Gain in db:
0
-50
-100
-150
-200
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalised frequency:
Phase spectrum
Phase in radians:
4
2
0
-2
-4
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Normalised frequency:

PART-B

Manual of DSP 6713 Starter Kit

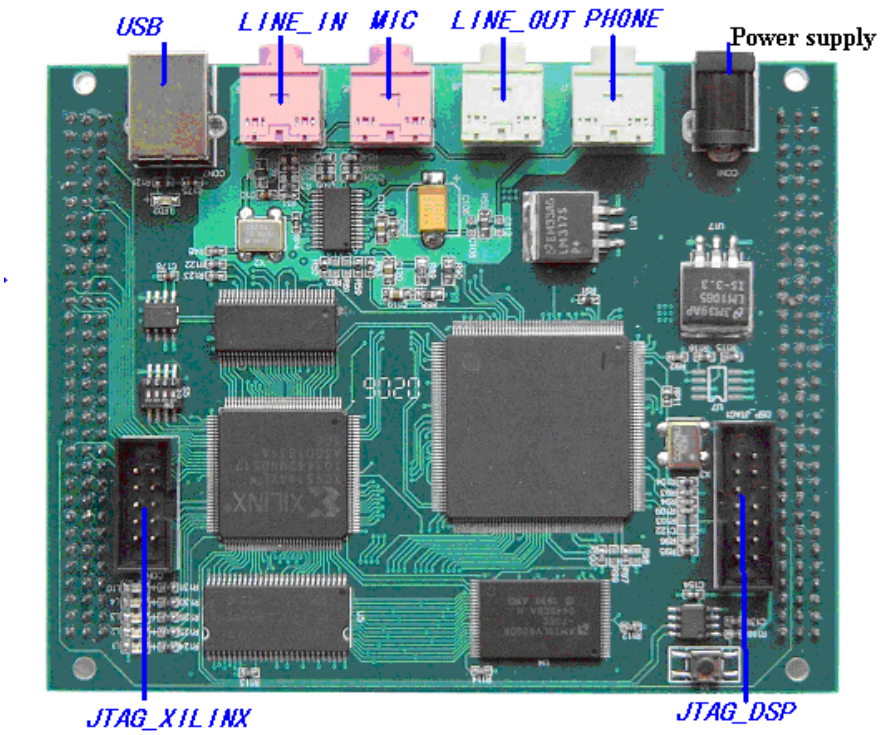
Contents

Consignment list

1. A main board of DSP6713 -1No
2. Power Supply 5V, 3A. - 1No
3. USB Programmer with USB Cable -1No
4. Head Phone with MIC -1No
5. Audio Jack -3No [1+2]
6. VTU Programs CD -1No
7 .5th SEM Lab DSP Manual -1No

Introduction of main board:

1. USB2.0 CY7C68013-56PVC, compatible with USB2.0 and USB1.1, including 8051
2. DSP TMS320C6713 TQFP-208 Package Device with, 4 layers board
3. SDRAM MT48LC4M16A2 1meg*16 *4 bank micron
4. FLASH AM29LV800B 8Mbit1Mbyte of AMD
5. RESET chip specialize for reset with button for manually reset
6. POWER power supply externally, special 5V, 3.3V, 1.6V chip for steady voltage with remaining for other devices.
7. EEPROM 24LC64 for download of USB firmware
8. CPLD XC95144XL
9. AIC TLV320AIC23B sampling with 8-96KHZ, 4 channels with interface of headphone.



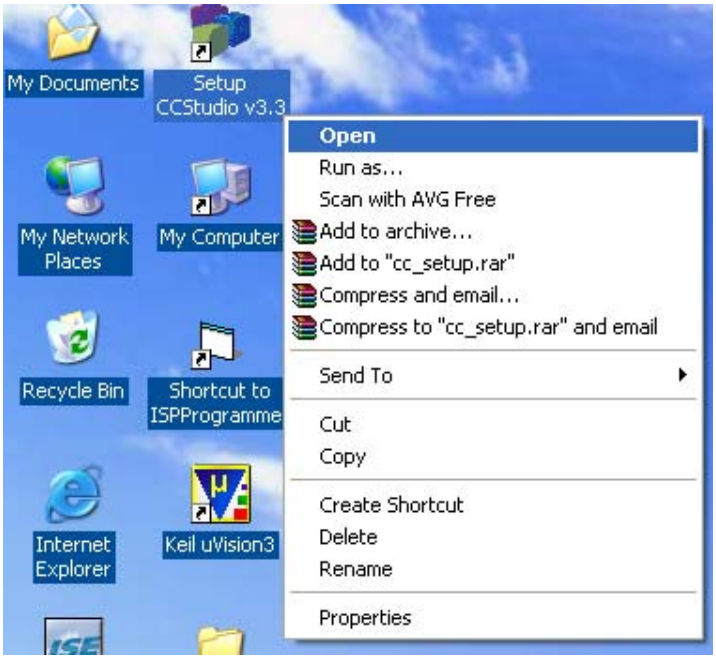
Board Picture:

The setup of USB Programmer and emulator in CCS v3.3

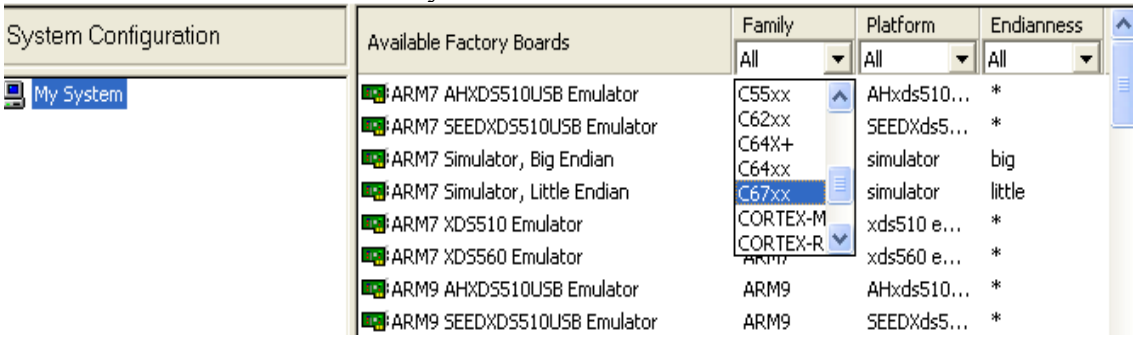
1. Install CCS v3.3 software according to the default Custom Install.
2. Install USB Emulator chooses to install directory from CCS v3.3 hat is if CCSv3.3 is installed in C: / CCStudio_v3.3 directory, then install the USB emulator driver in this directory.

Procedure to Setup Emulator:

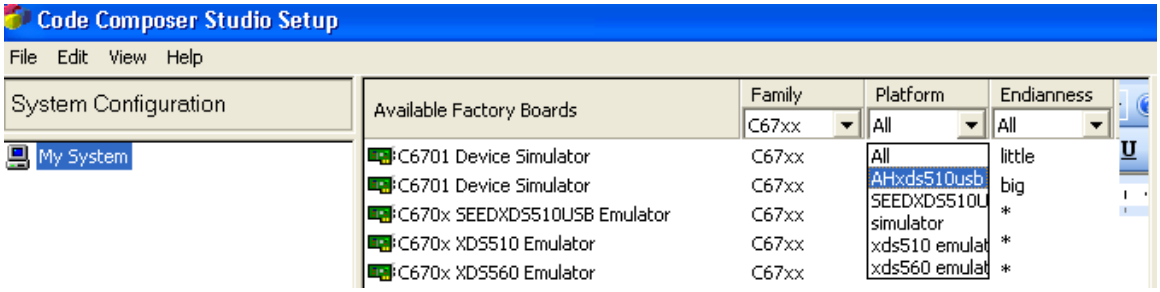
3. Open the “Setup CCStudio v3.3”



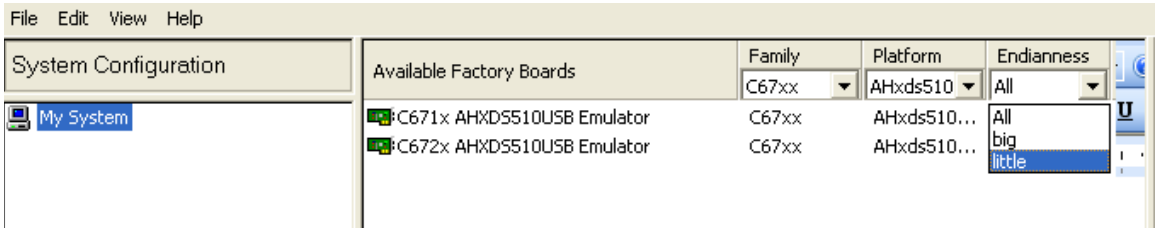
4. Choose c67xx in the “Family”.



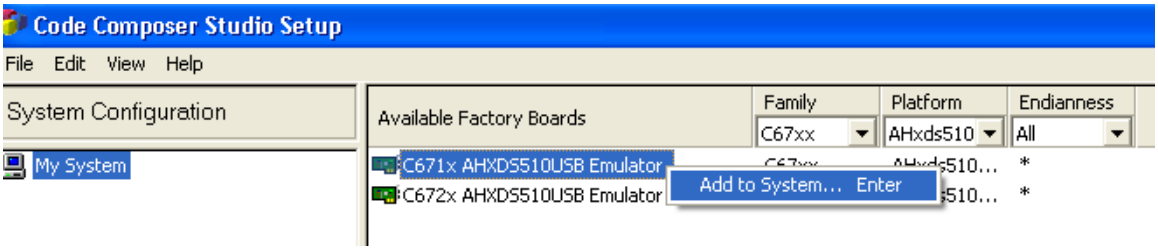
5. Choose AHxds510usb emulator in the “Platform”.



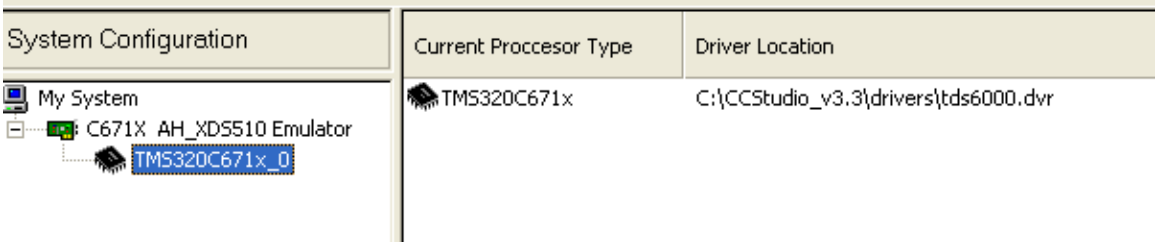
6. Choose little in the “Endianness”.



7 Now you are left with two options under Available Factory Boards, Choose C671X AHXDS510 USB Emulator, right click and “Add to system…”



8. Now the Emulator and the processor both are selected under “system configuration”.

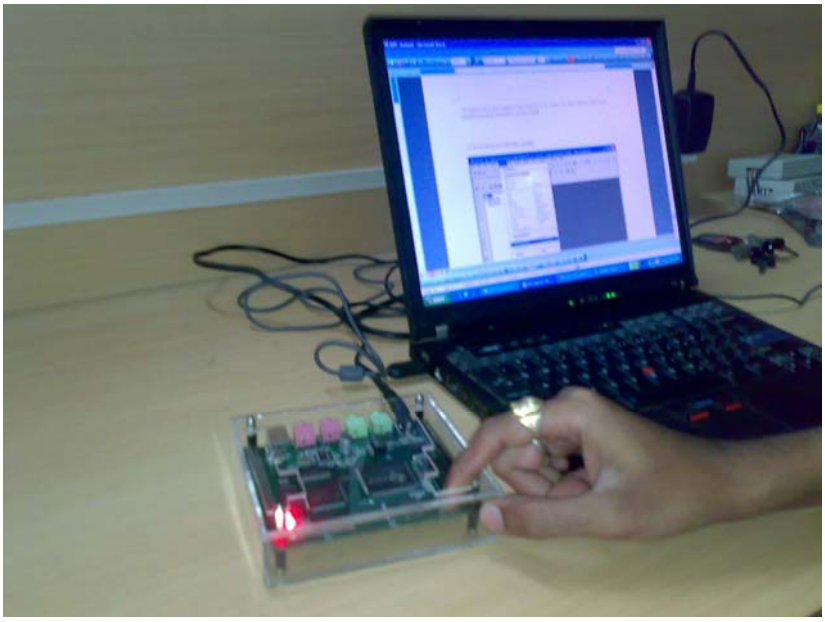


9 Choose file and click on “save”.
10 Choose file and click on “exit” you will get a wizard as shown bellow

Click on yes.

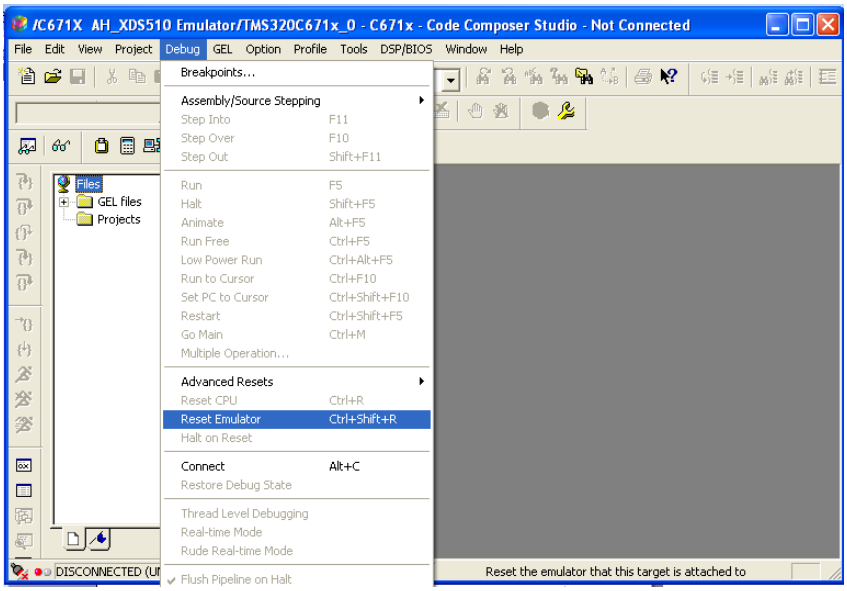
Then it launches a Code Composer Studio

11 Connect the power supply to the board (5V, 3A): make sure that supply is there in the board by pressing reset button.

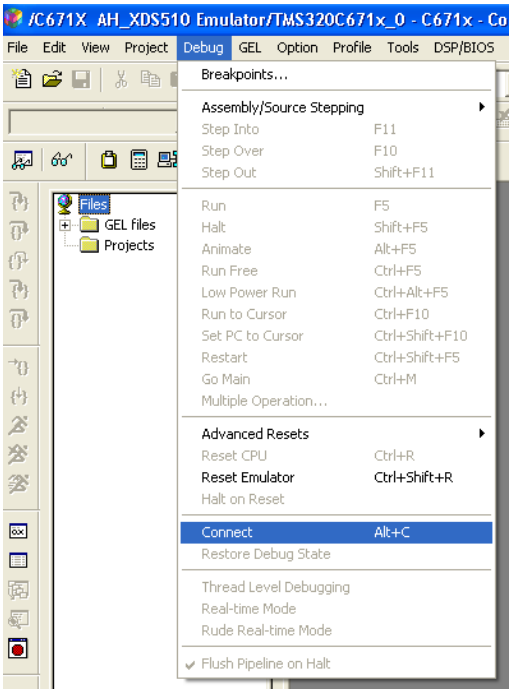


11 After 2 minutes after power supply is turned on then connects the USB Programmer cable to DSP JTAG connector and Host Computer where CCS 3.3 is installed.

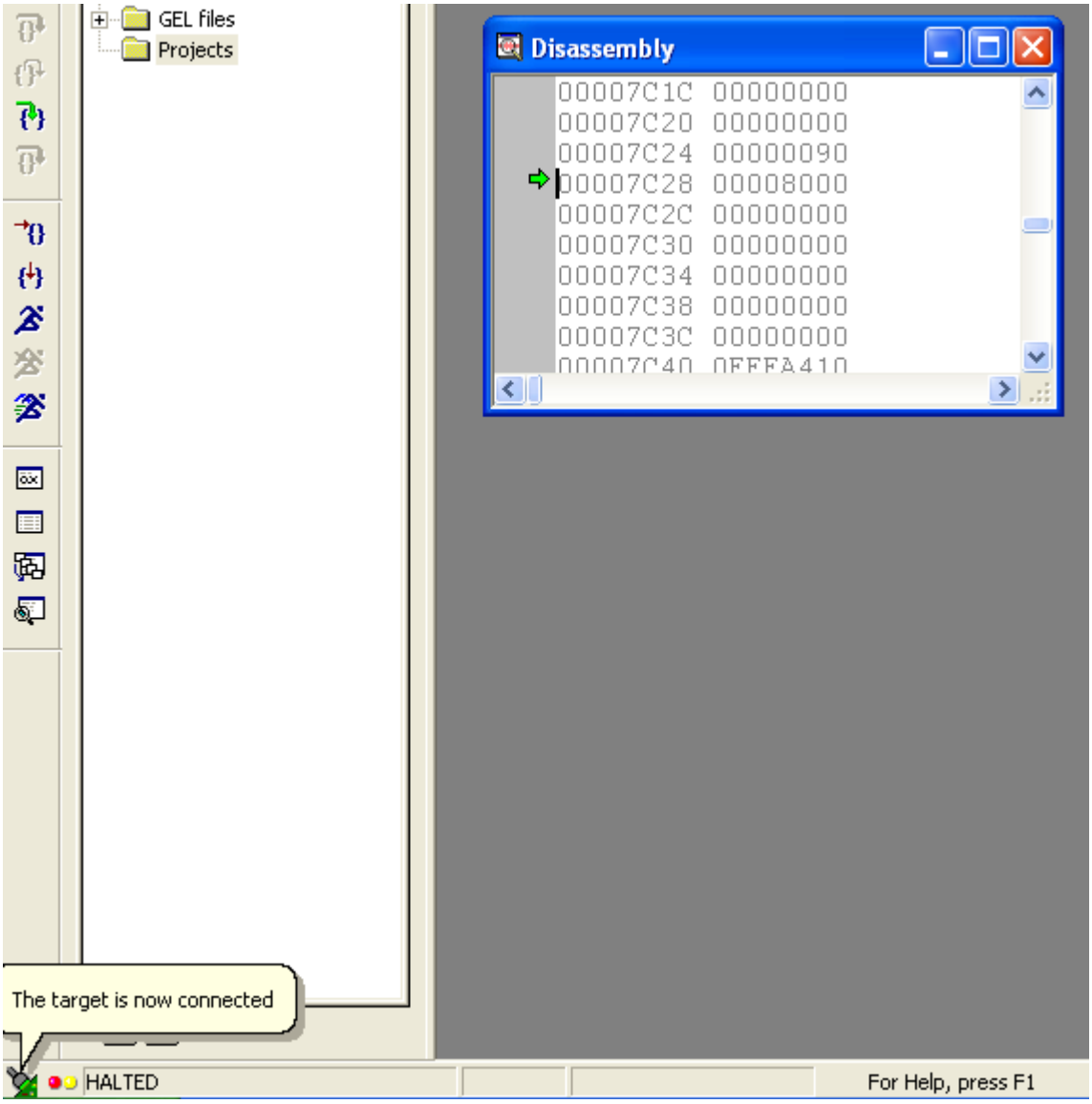
12. Go to Debug and select the Reset Emulator



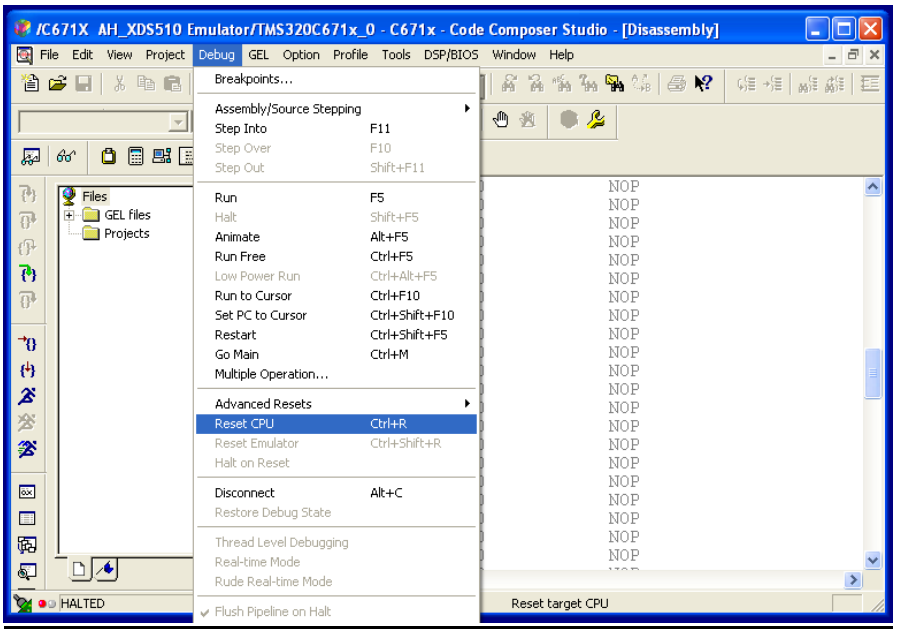
13. Go to Debug and select the option connect by Pressing Reset Button on the board



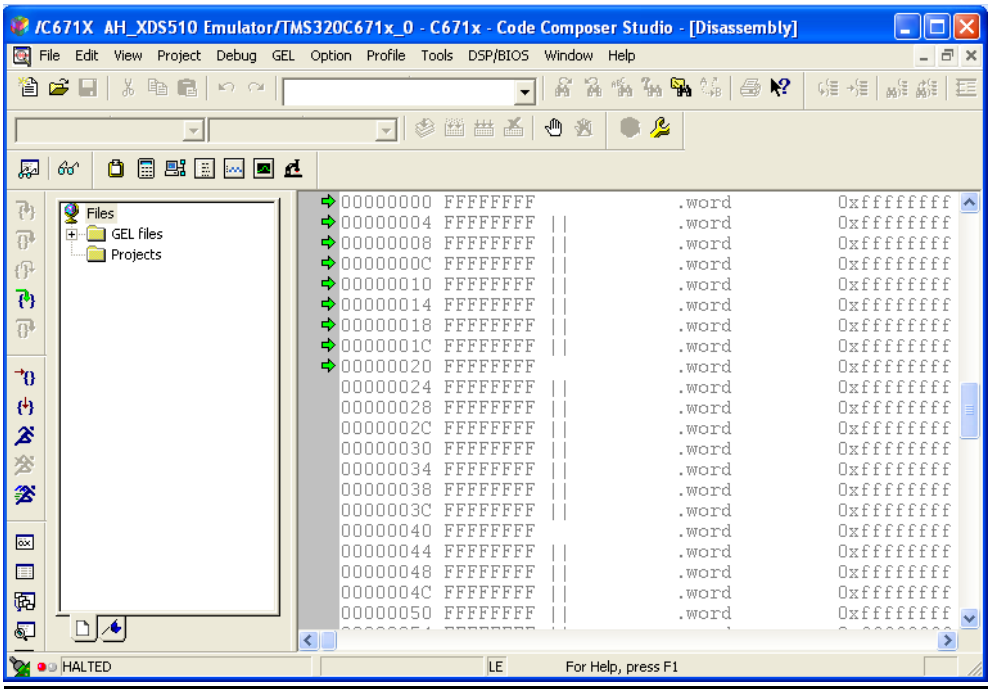
14. Now Target is connected



15 Go to Debug and press Reset CPU that will initialize your memory.



After Pressing Reset the CPU following window should appear in CCS.

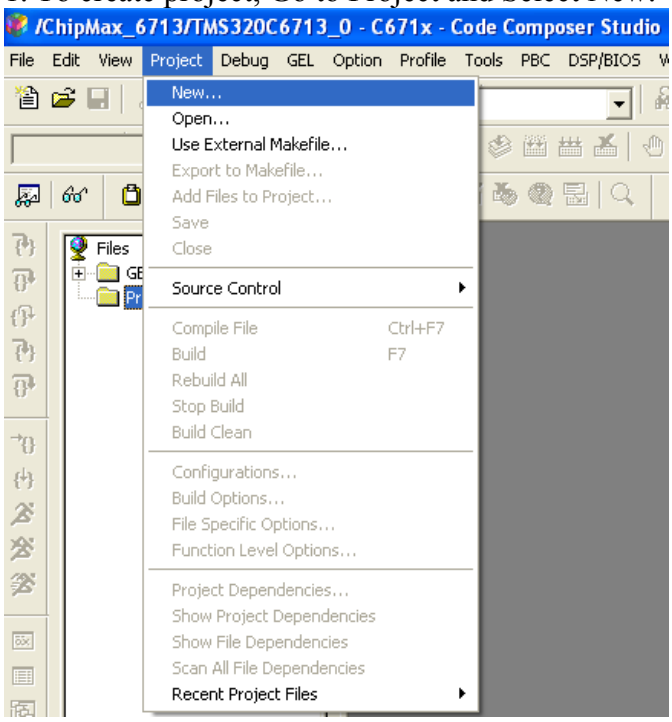


Now Board is ready for working real and non real time programs.

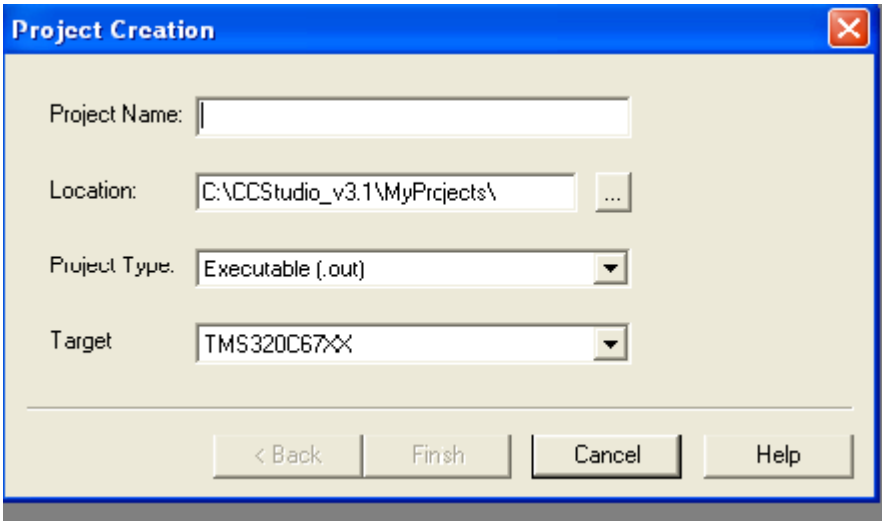
Experiment 1: Linear Convolution

Procedure to create new Project:

1. To create project, Go to Project and Select New.

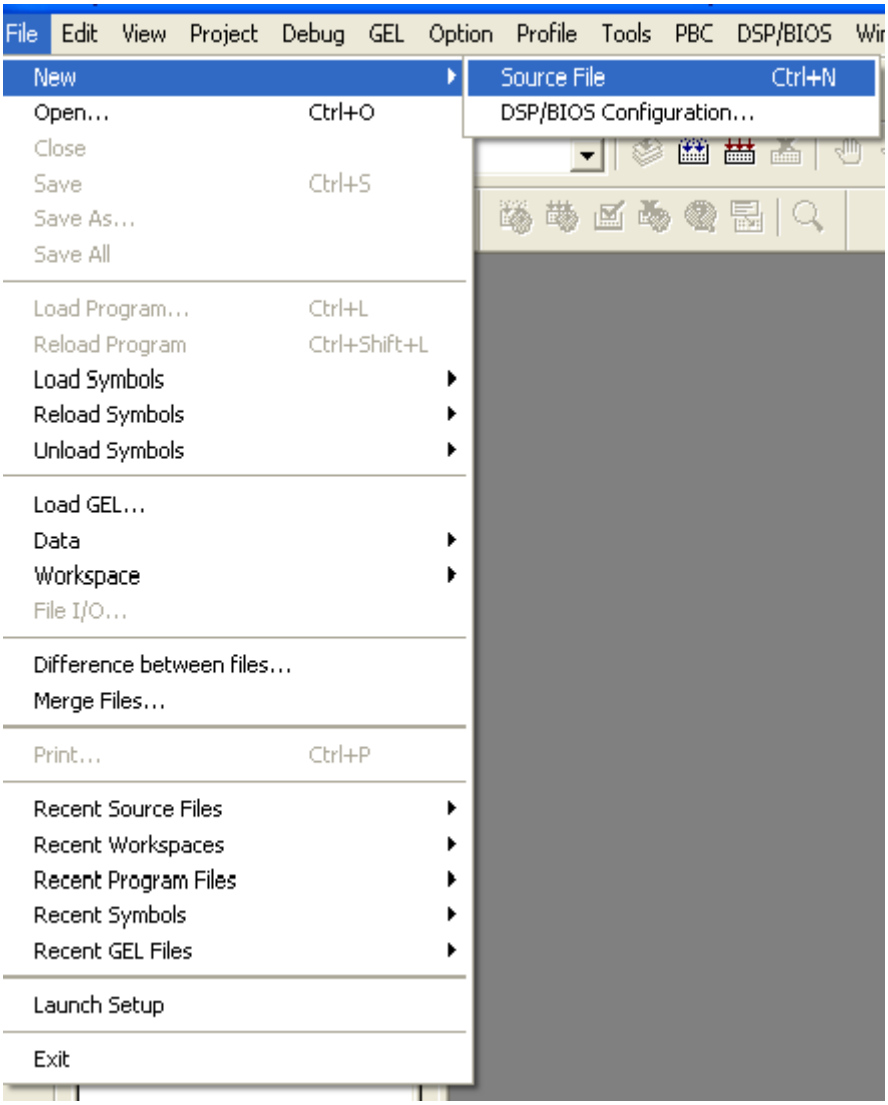


2. Give project name and click on finish.



(Note**:** Location must be c:\CCStudio_v3.3\MyProjects).

3. Click on File ⇨ New ⇨ Source File, To write the Source Code.



Aim: Linear Convolution of the two given sequences

Mathematical Formula:

The linear convolution of two continuous time signals x(t) and h(t) is defined by

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau)\, d\tau$$

For discrete time signals x(n) and h(n), is defined by

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)\, h(n-k)$$

Where x(n) is the input signal and h(n) is the impulse response of the system.

In linear convolution length of output sequence is,
Length (y(n)) = length(x(n)) + length(h(n)) – 1

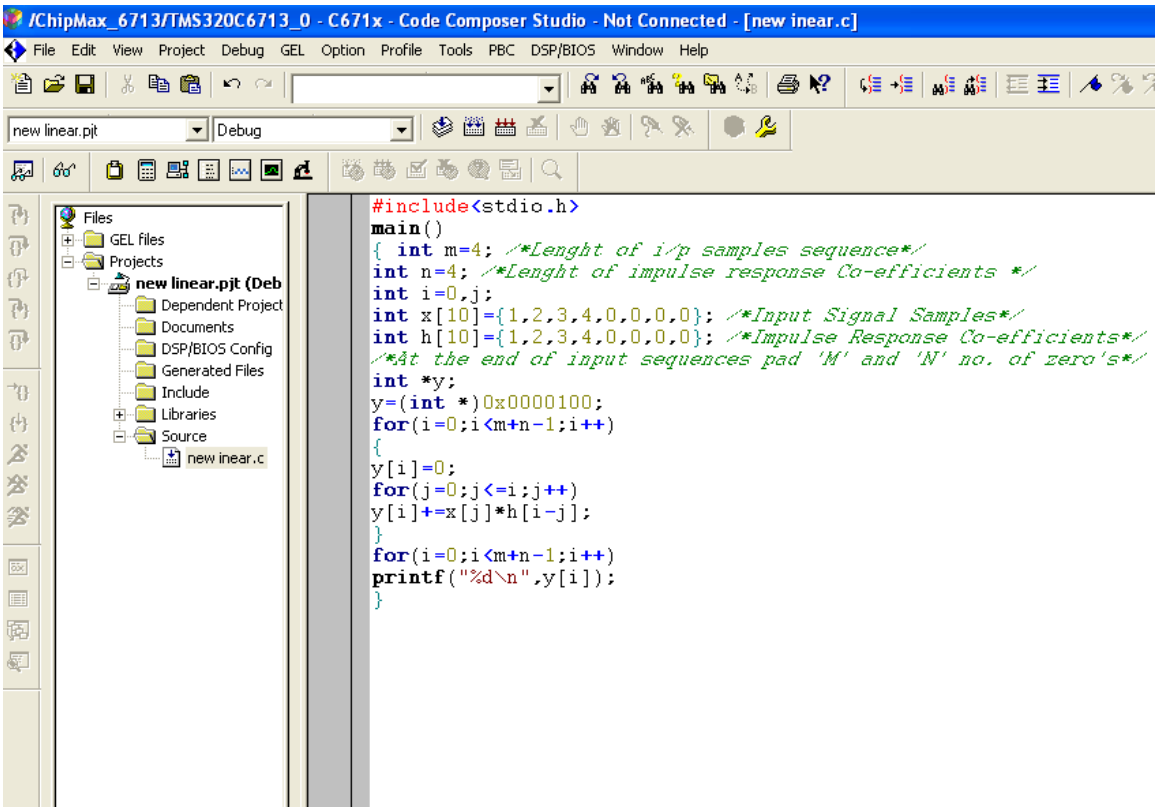
Program:

```
#include<stdio.h>
main()
{
int m=4;                        /*Lengh of i/p samples sequence*/
int n=4;                        /*Lengh of impulse response Co-efficients */
int i=0,j;
int x[10]={1,2,3,4,0,0,0,0};    /*Input Signal Samples*/
int h[10]={1,2,3,4,0,0,0,0};    /*Impulse Response Co-efficients*/
int *y;
y=(int *)0x0000100;
for(i=0;i<m+n-1;i++)
        {
        y[i]=0;
        for(j=0;j<=i;j++)
        y[i]+=x[j]*h[i-j];
        }
for(i=0;i<m+n-1;i++)
        printf("%d      ",y[i]);
}
```

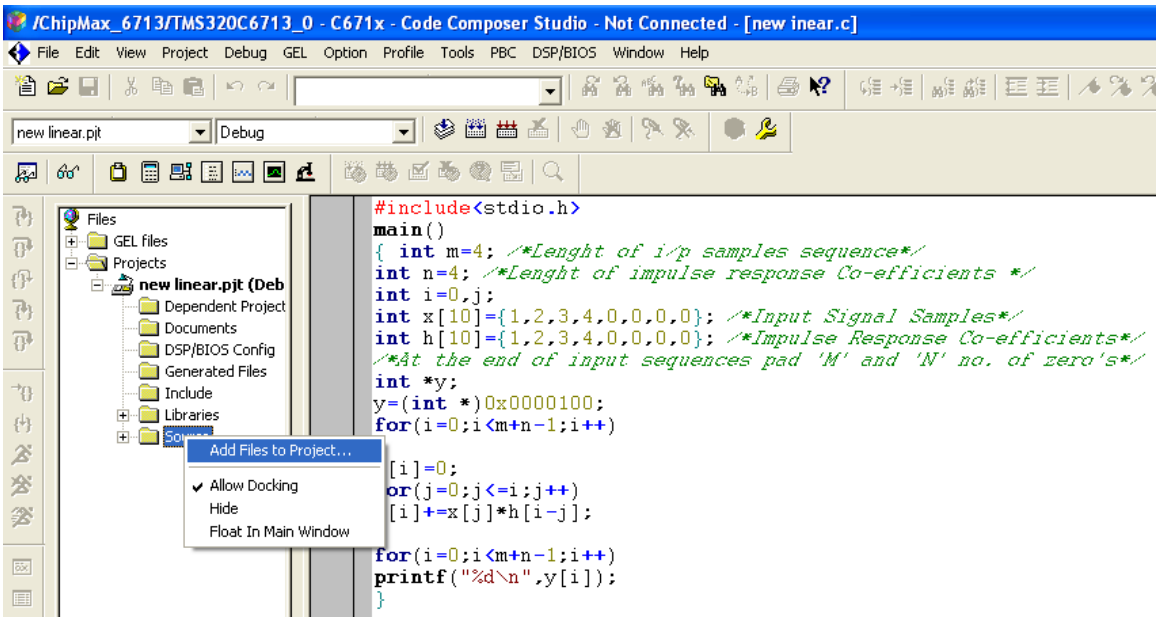
Output:

1, 4, 10, 20, 25, 24, 16.

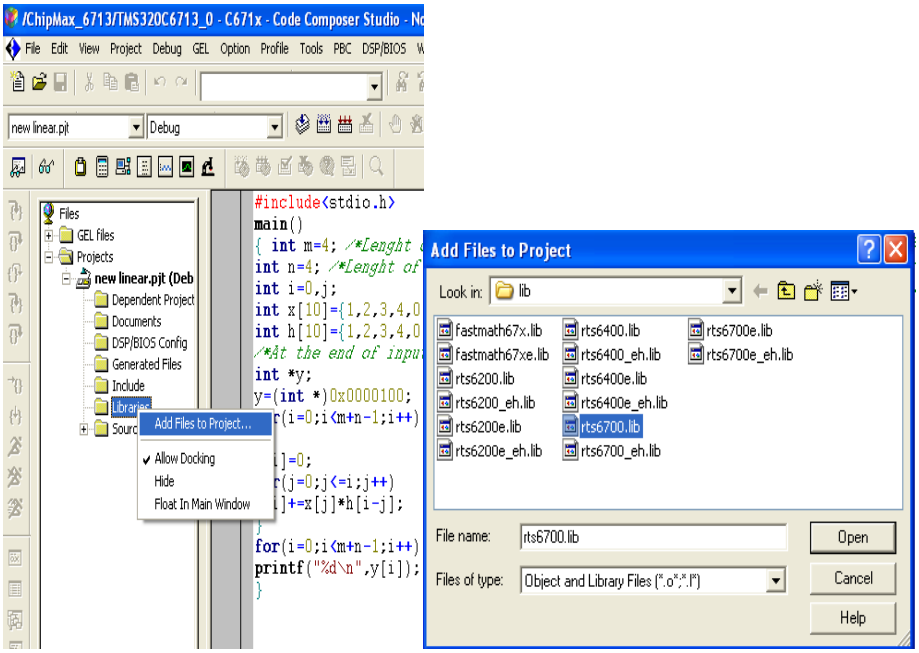
4. Enter the source code and save the file with “.C” extension.



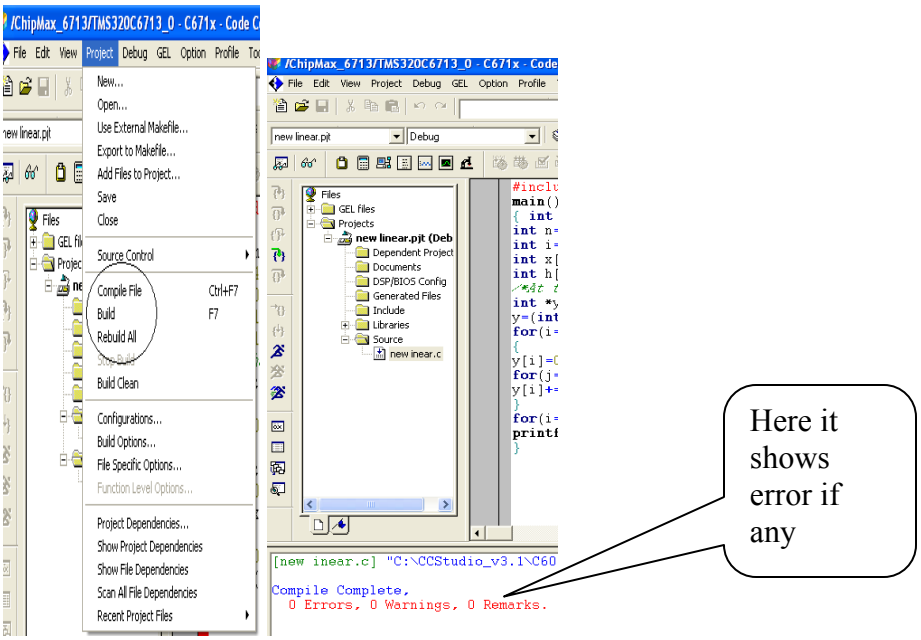
5. Right click on source, Select add files to project .. and Choose “.C “ file Saved before.



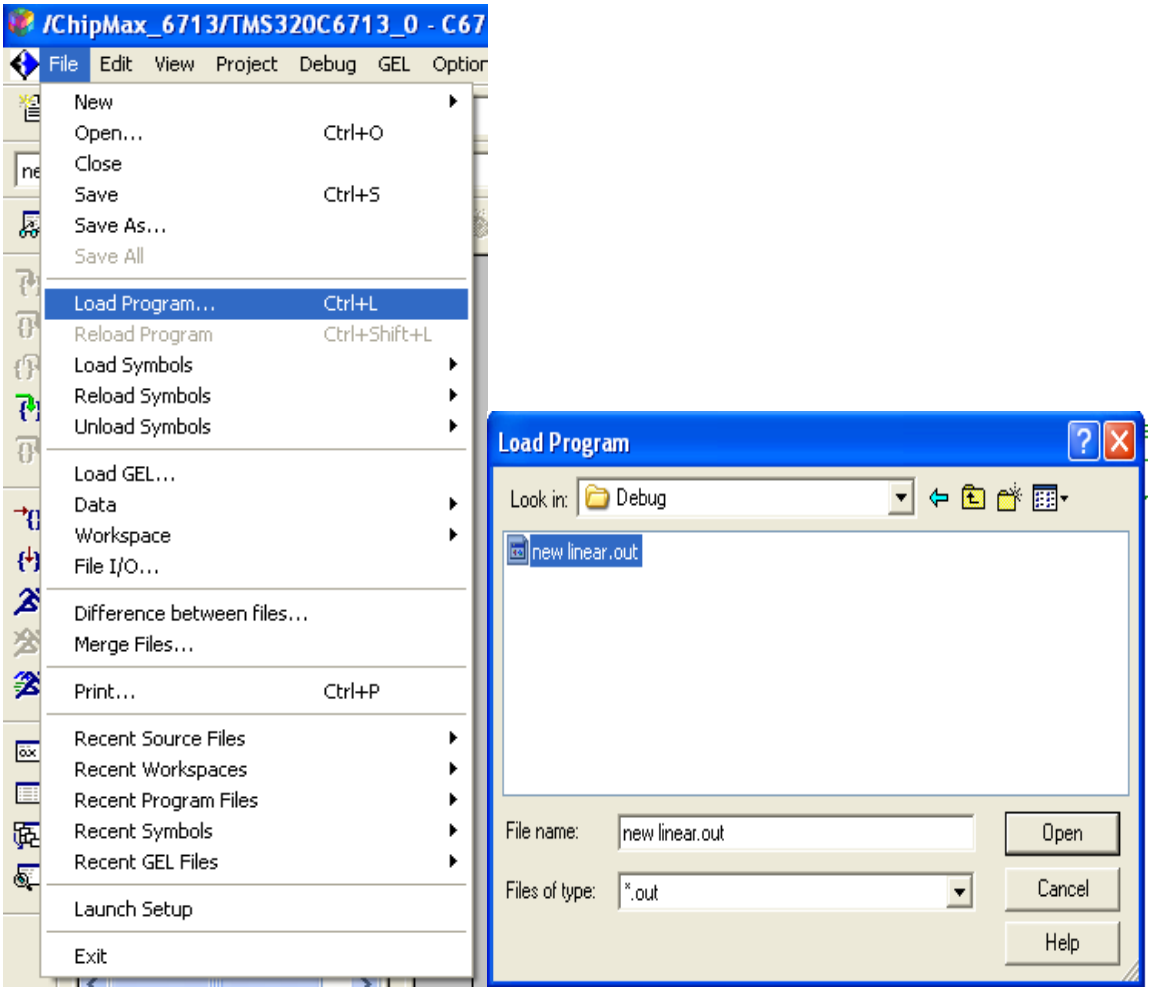
6. Right Click on libraries and select add files to Project.. and choose C:\CCStudio_v3.3\C6000\cgtools\lib\rts6700.lib and click open.



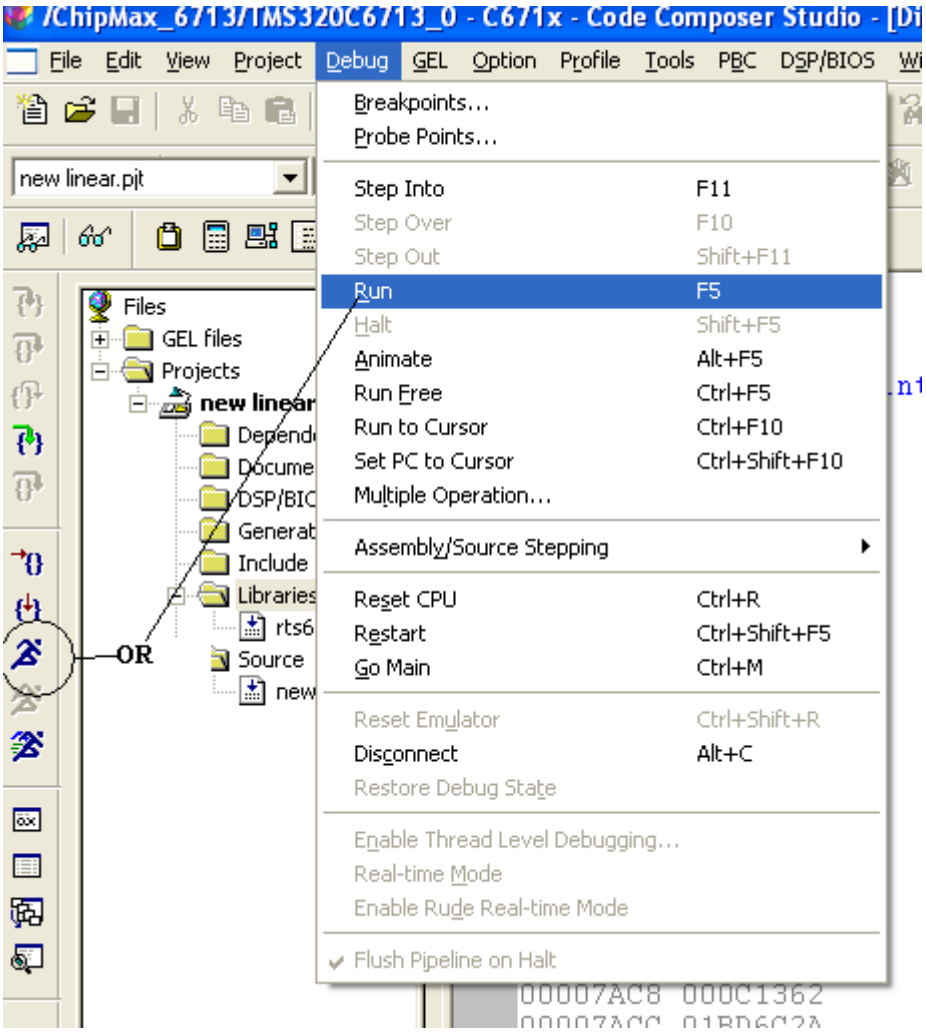
7. a)Go to Project to Compile .
   b) Go to Project to Build.
   c) Go to Project to Rebuild All.



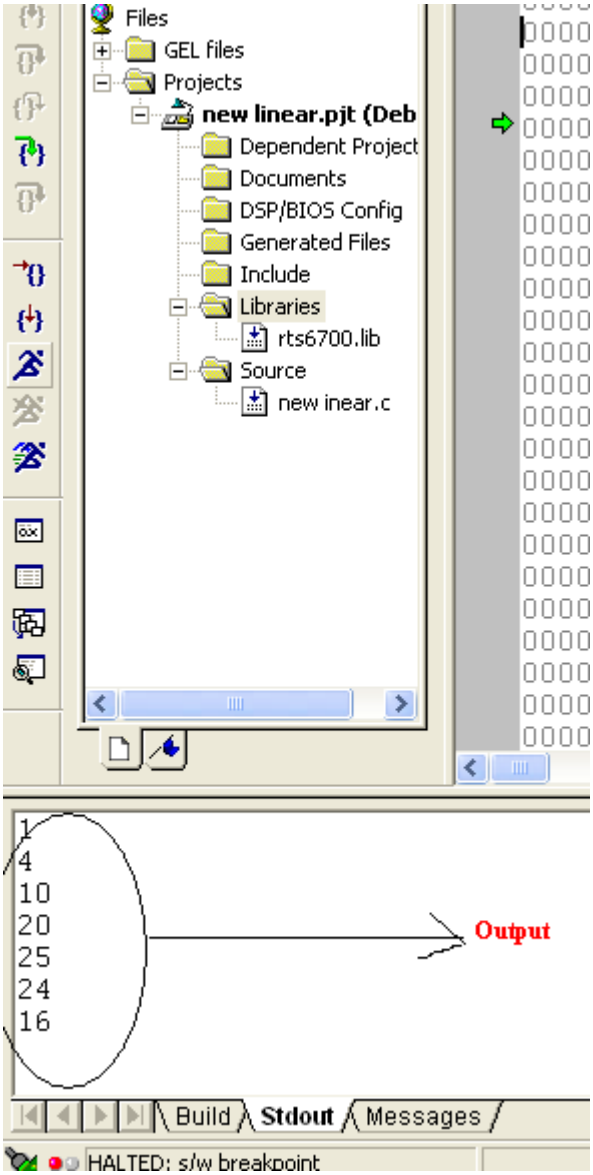
8. Go to file and load program and load **“.out”** file into the board..



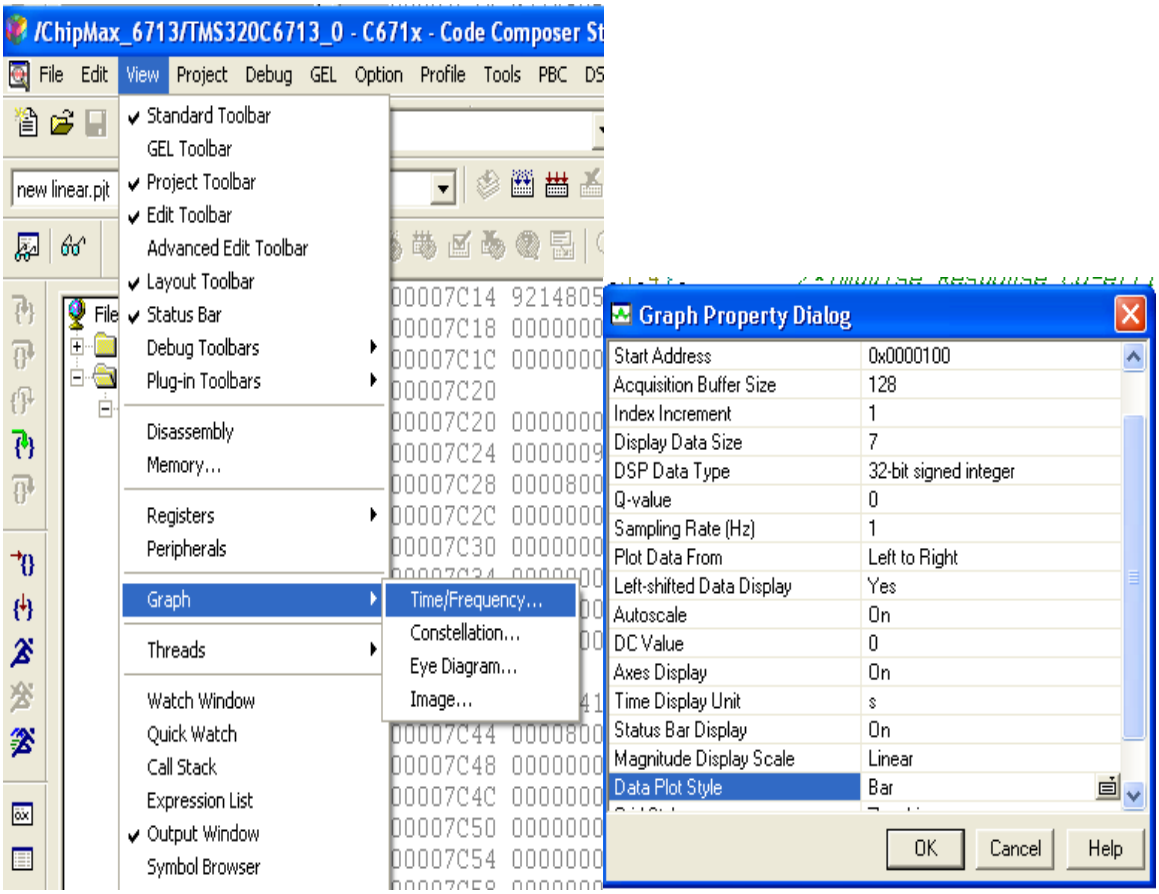
9. Go to Debug and click on run to run the program.



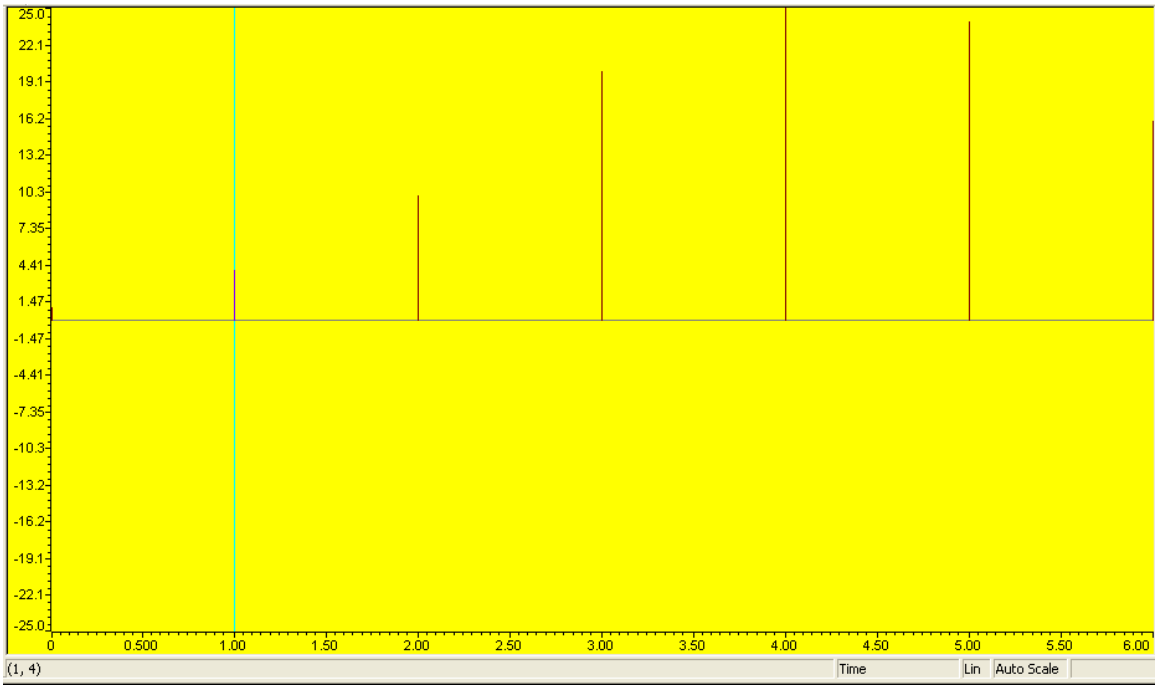
10. Observe the output in output window.



11. To see the Graph go to View and select time/frequency in the Graph, And give the correct Start address provided in the program, Display data can be taken as per user.



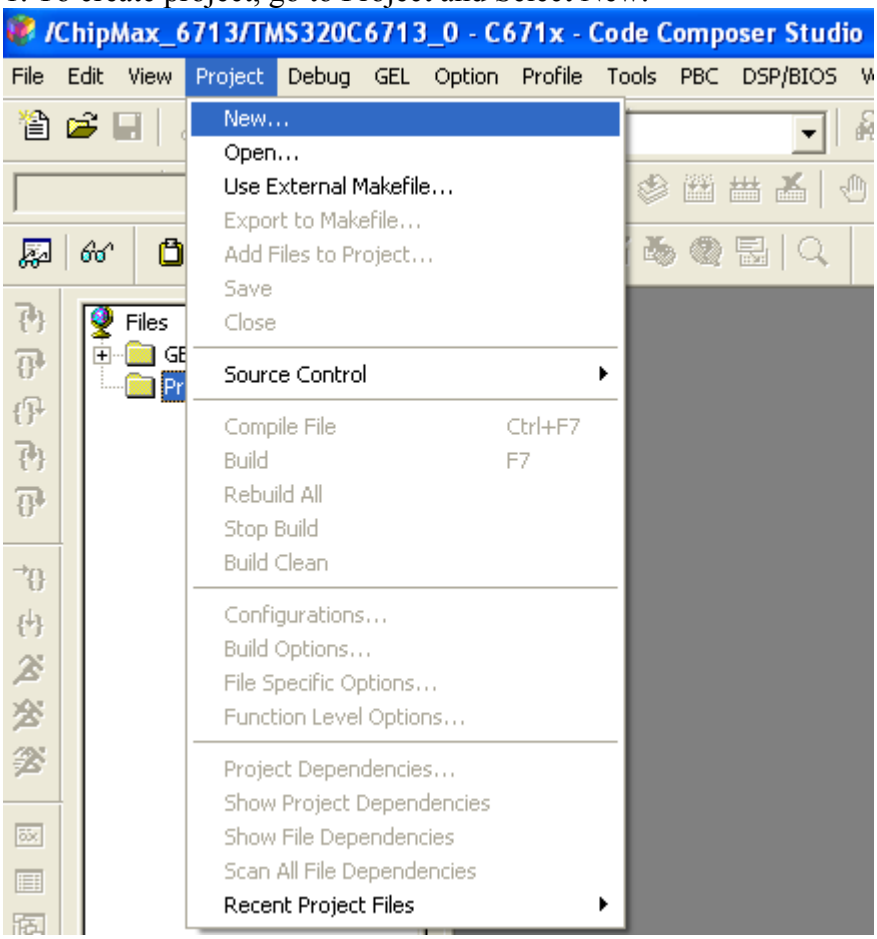
12. Green line is to choose the point, Value at the point can be seen (Highlighted by circle at the left corner).



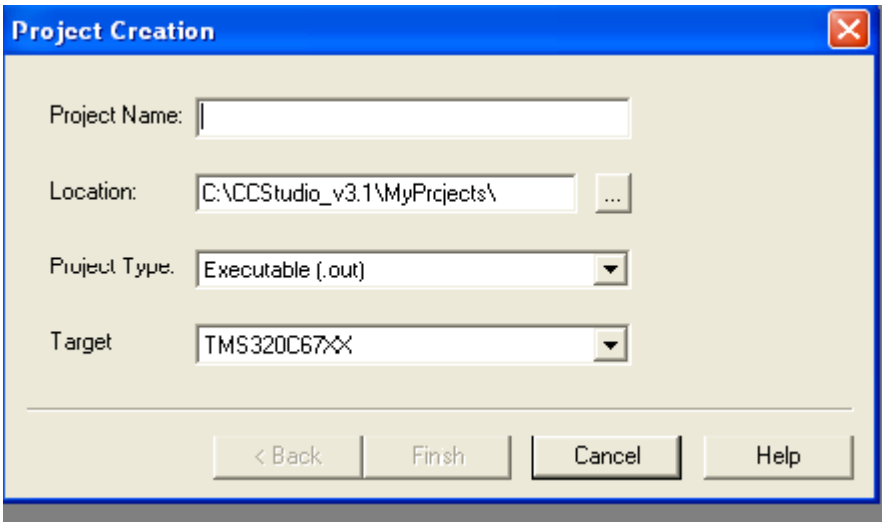
Experiment 2: Circular Convolution

Procedure to create new Project:

1. To create project, go to Project and Select New.

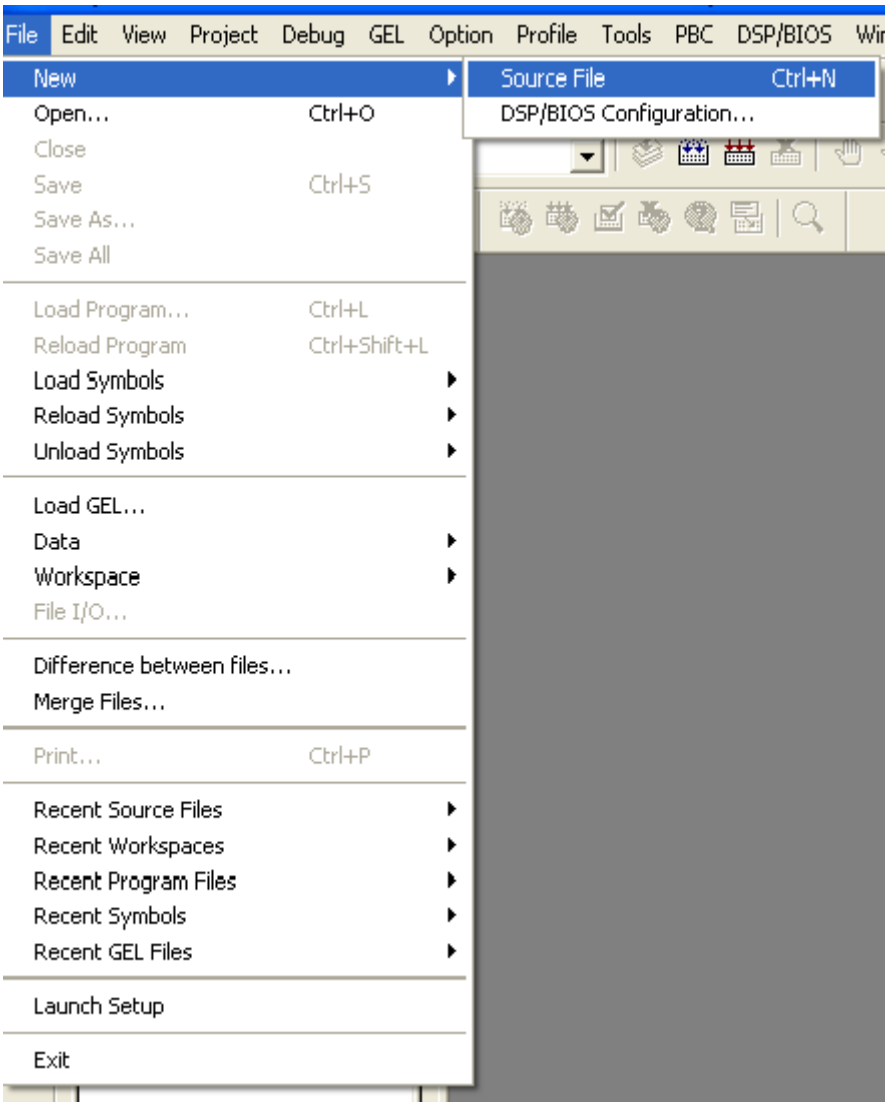


2. Give project name and click on finish.



(**Note:** Location must be c:\CCStudio_v3.3\MyProjects).

3. Click on File ⇨ New ⇨ Source File, to write the Source Code.



AIM: To implement circular convolution of two sequences.

Circular Convolution:

Let $x_1(n)$ and $x_2(n)$ are finite duration sequences both of length N with DFT's $X_1(k)$ and $X_2(k)$. Convolution of two given sequences $x_1(n)$ and $x_2(n)$ is given by the equation,

$x_3(n) = IDFT[X_3(k)]$

$X_3(k) = X_1(k)\ X_2(k)$

$$x_3(n) = \sum_{m=0}^{N-1} x_1(m)\ x_2((n-m))_N$$

Program:

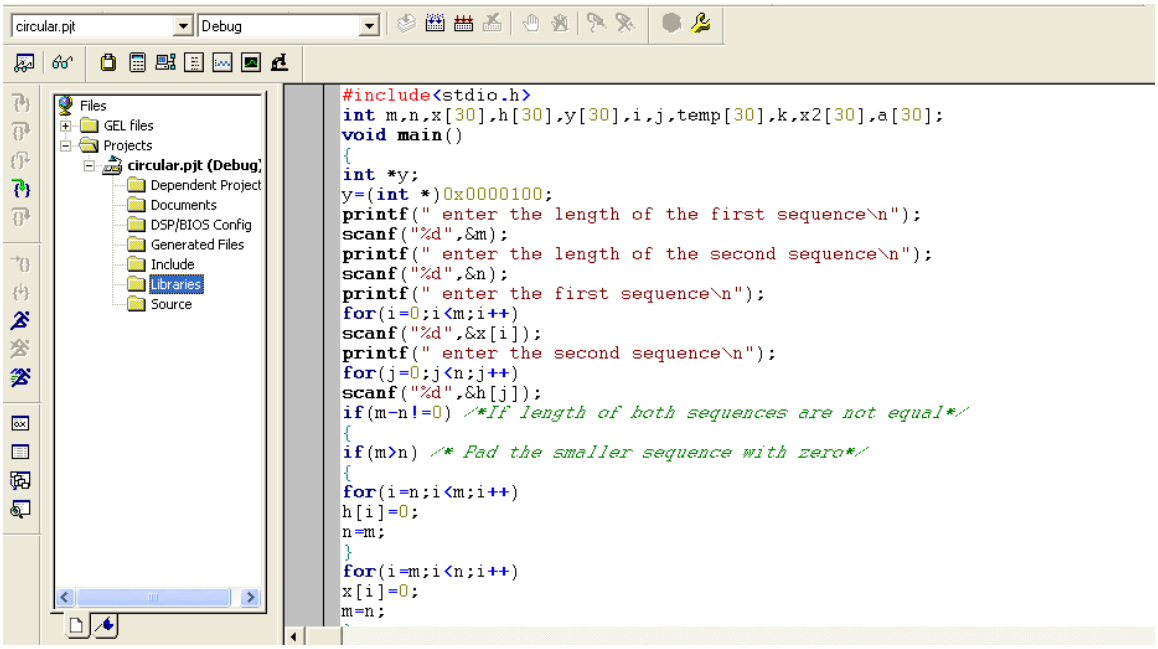
```
#include<stdio.h>
int m,n,x[30],h[30],y[30],i,j,k,x2[30],a[30];
void main()
{
printf("Enter the length of the first sequence:\n");
scanf("%d",&m);
printf("Enter the first sequence:\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf("Enter the length of the second sequence:\n");
scanf("%d",&n);
printf("Enter the second sequence:\n");
for(j=0;j<n;j++)
scanf("%d",&h[j]);

/*If length of both sequences are not equal*/
if(m-n!=0)
    {
     if(m>n) /* Pad the smaller sequence with zero*/
          {
          for(i=n;i<m;i++)
               h[i]=0;
          n=m;
          }
     for(i=m;i<n;i++)
          x[i]=0;
     m=n;
    }
y[0]=0;
```

```
a[0]=h[0];
for(j=1;j<n;j++) /*folding h(n) to h(-n)*/
a[j]=h[n-j];
/*Circular convolution*/
for(i=0;i<n;i++)
y[0]+=x[i]*a[i];
for(k=1;k<n;k++)
  {
    y[k]=0;
/*circular shift*/
for(j=1;j<n;j++)
     x2[j]=a[j-1];

x2[0]=a[n-1];
for(i=0;i<n;i++)
    {
     a[i]=x2[i];
     y[k]+=x[i]*x2[i];
    }
  }
/*displaying the result*/
printf(" the circular convolution is\n");
for(i=0;i<n;i++)
     printf("%d ",y[i]);
}
```

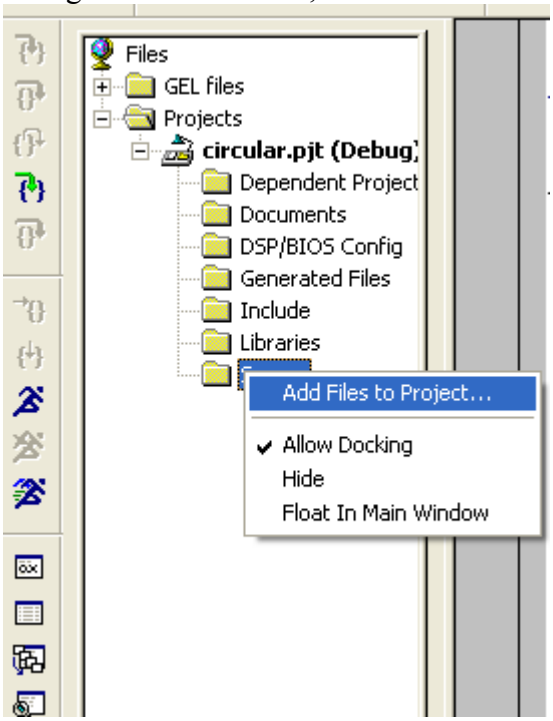
Output:

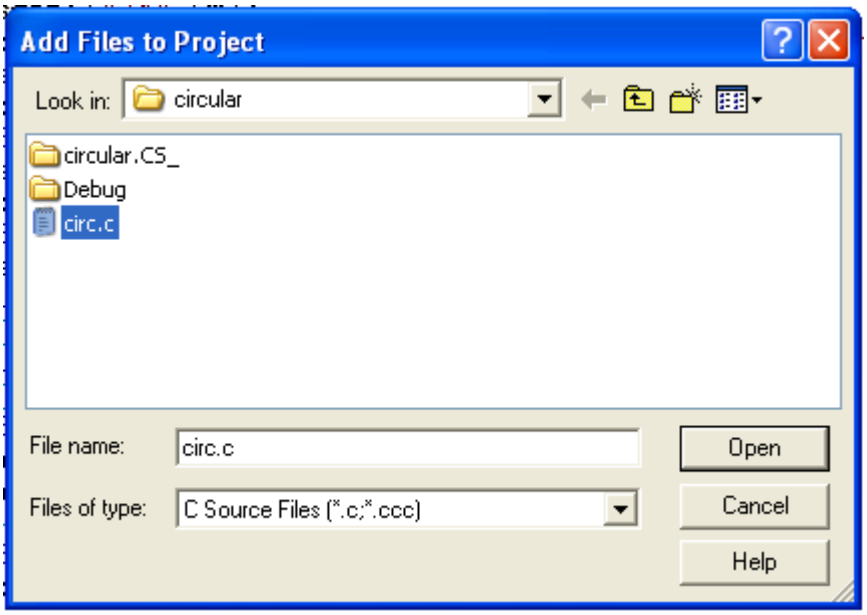
```
enter the length of the first sequence
4
 enter the length of the second sequence
4
 enter the first sequence
4 3 2 1
 enter the second sequence
1 1 1 1
 the circular convolution is
10 10 10 10
```

4. Enter the source code and save the file with “.C” extension.

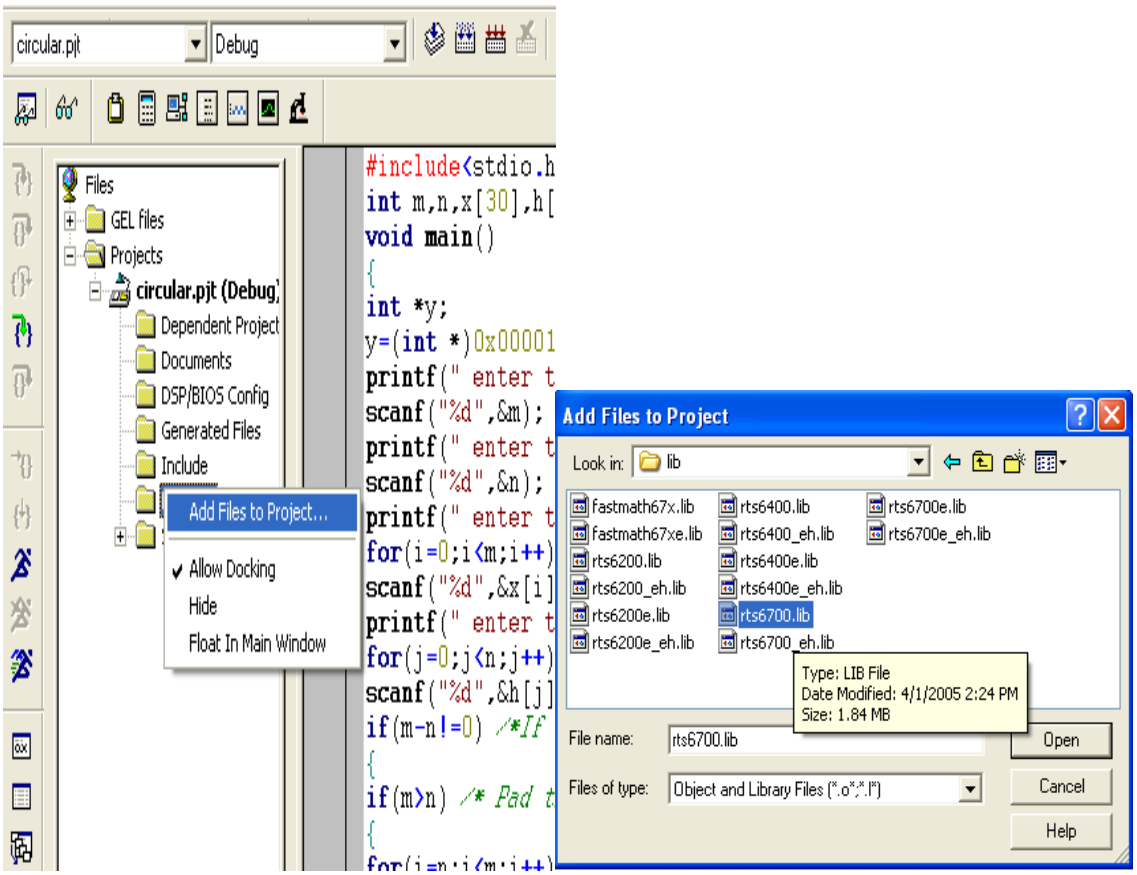


5. Right click on source, Select add files to project .. and Choose “.C “ file Saved before.

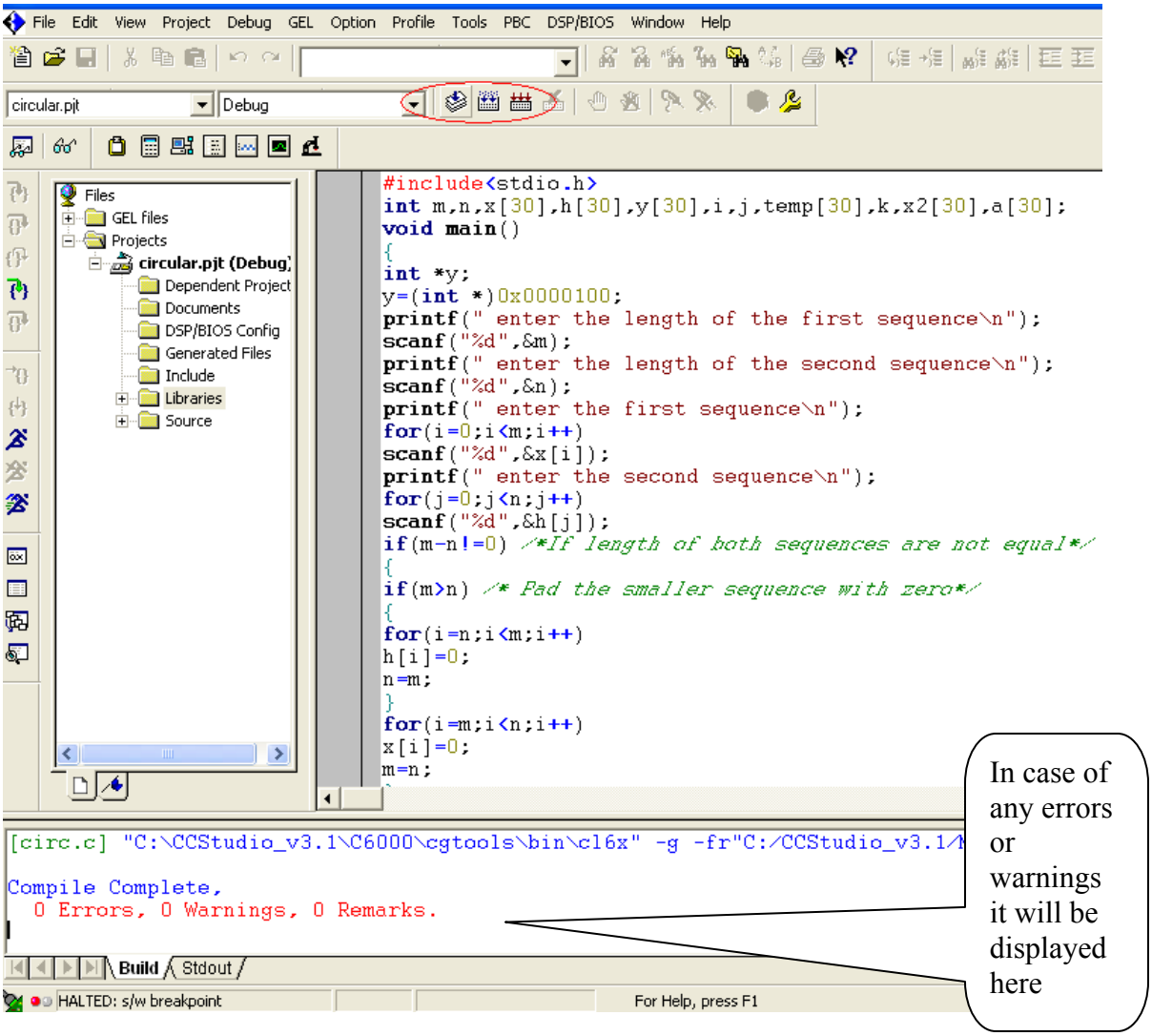




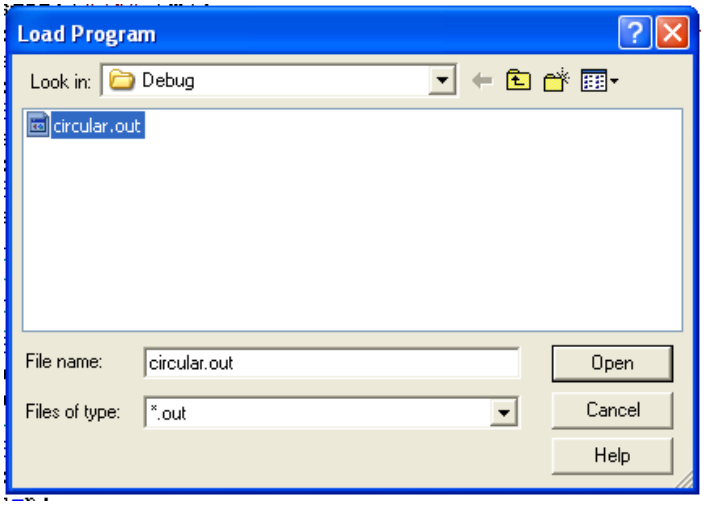
6. Right Click on libraries and select add files to Project.. and choose C:\CCStudio_v3.3\C6000\cgtools\lib\rts6700.lib and click open.



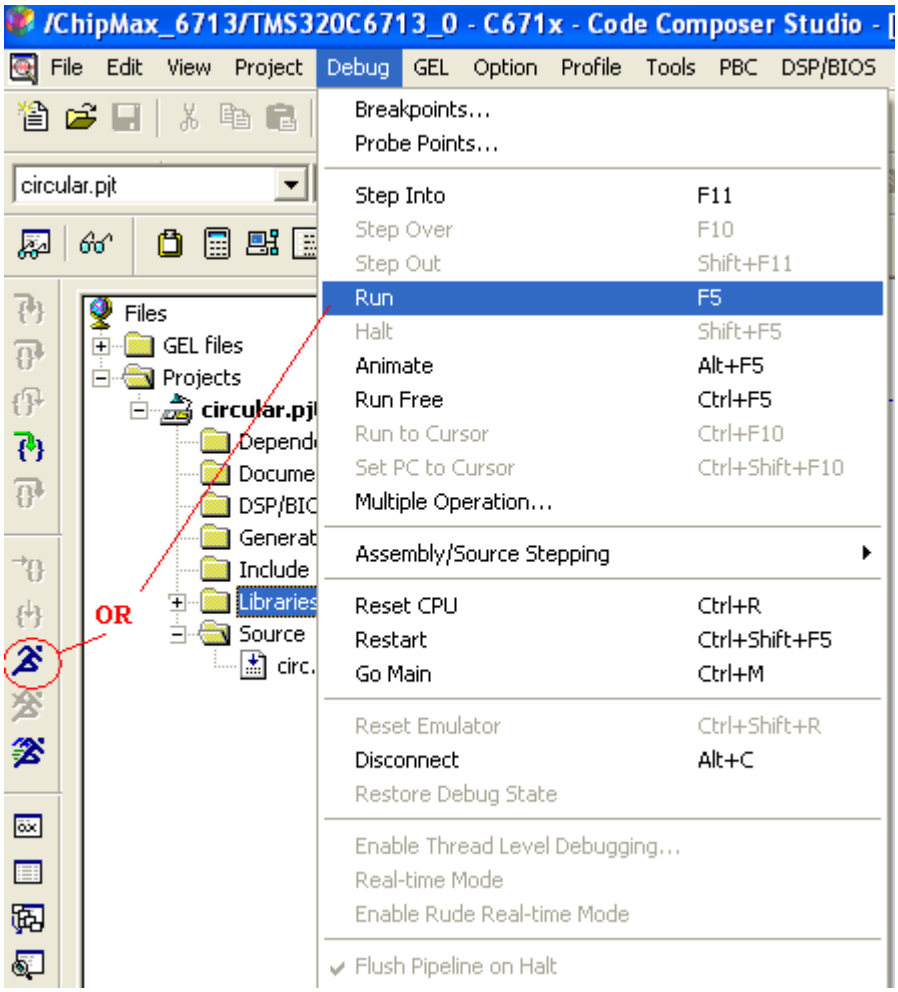
7. a) Go to Project to Compile .
   b) Go to Project to Build.
   c) Go to Project to Rebuild All.



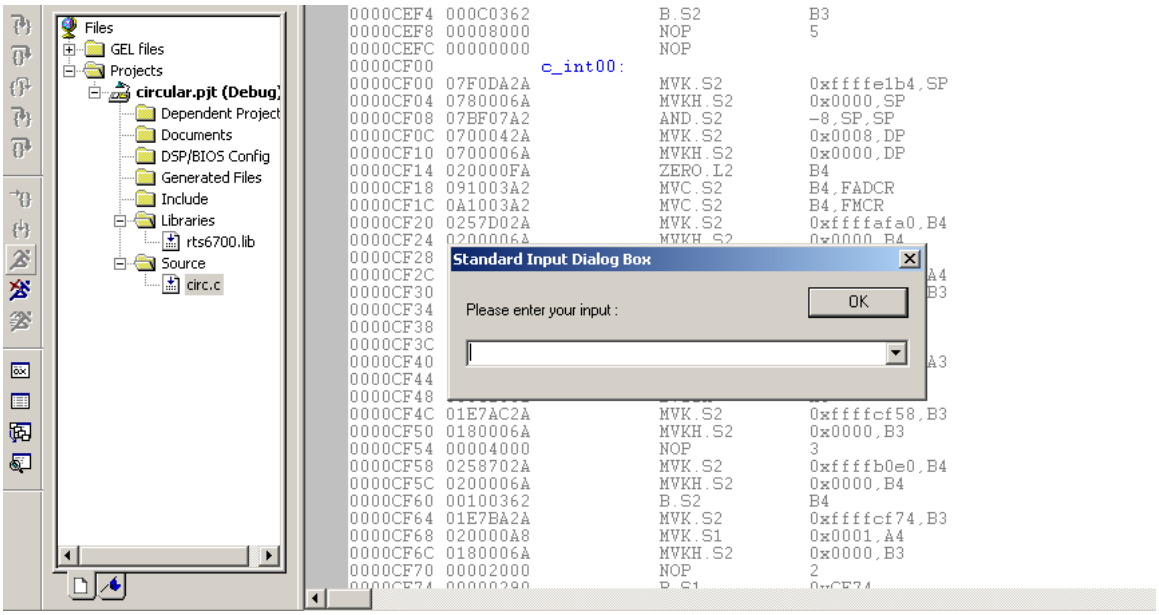
8. Go to file and load program and load **".out"** file into the board..

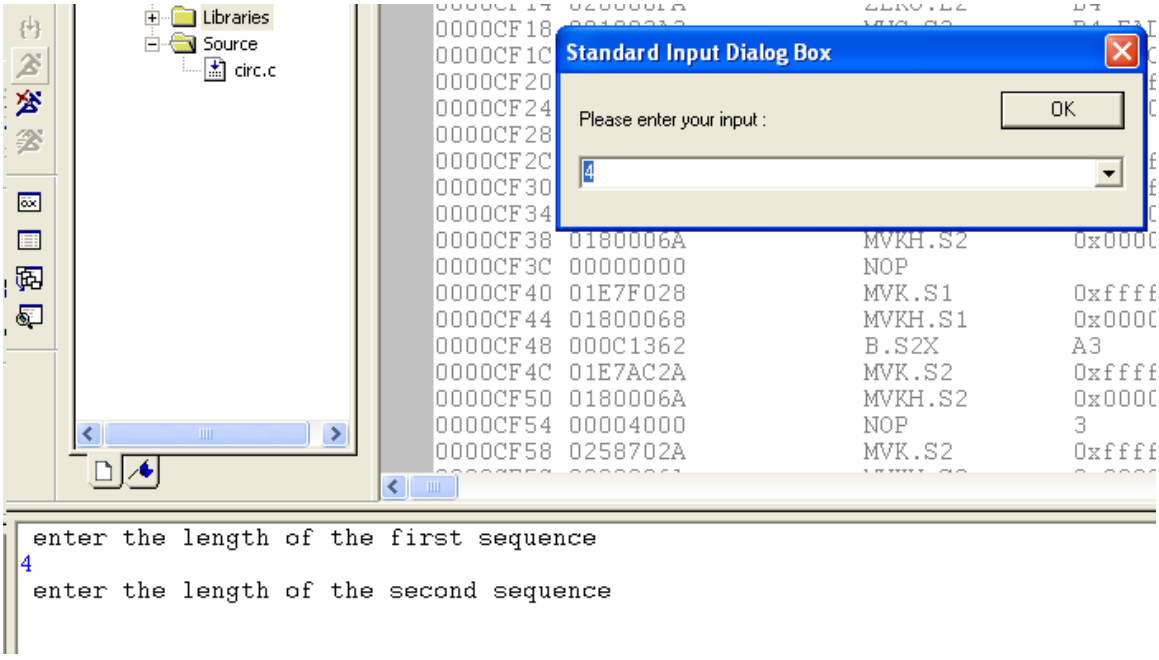


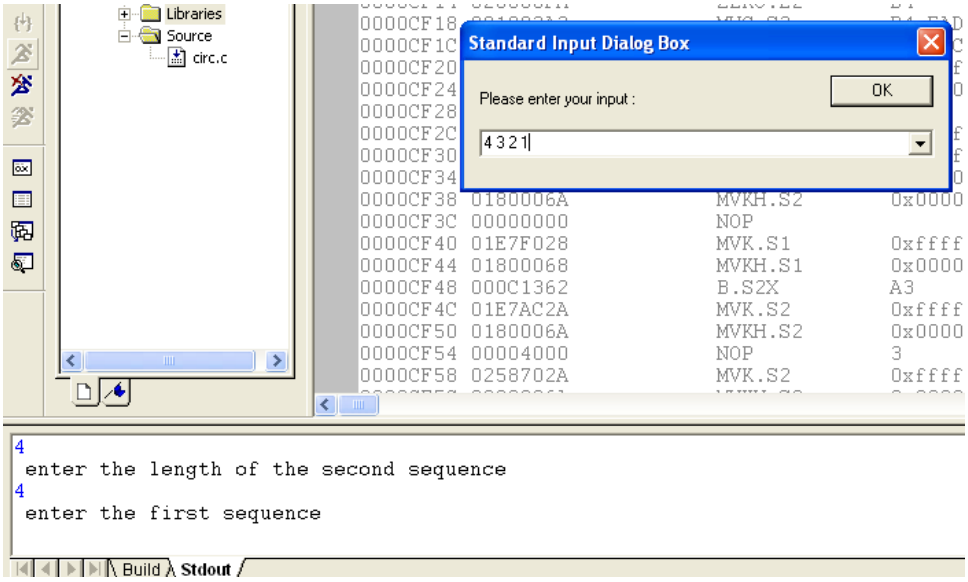
9. Go to Debug and click on run to run the program.

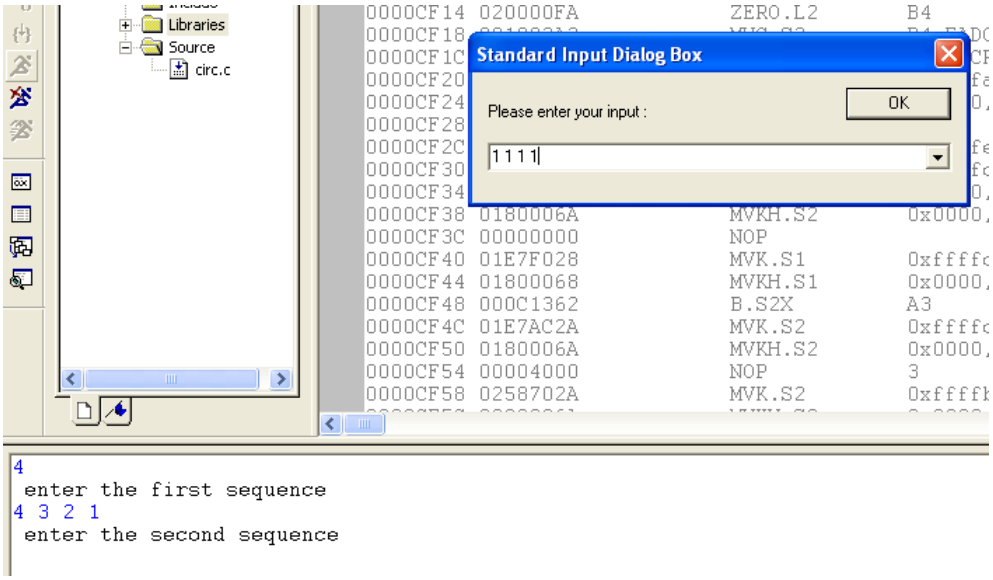


10. Enter the input data to calculate the circular convolution.

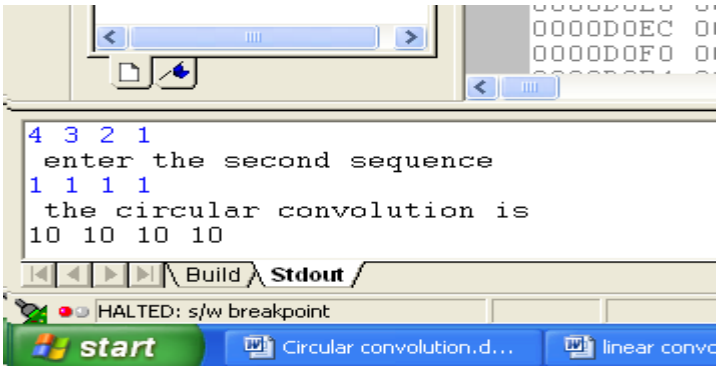




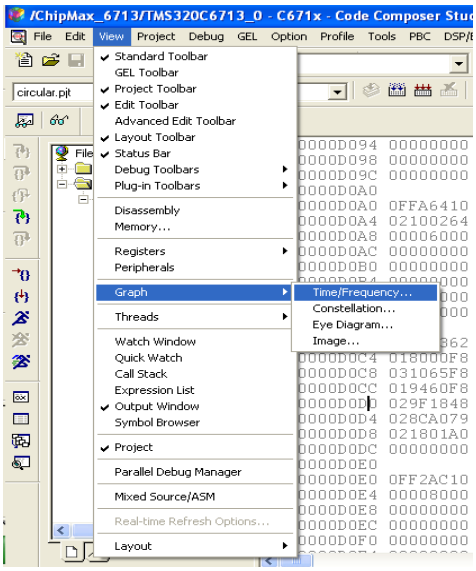


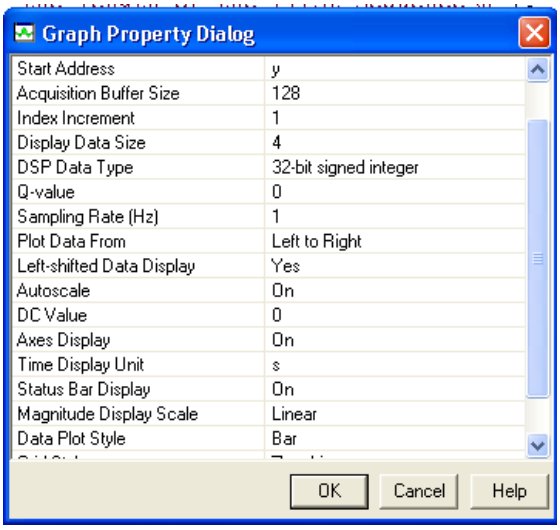


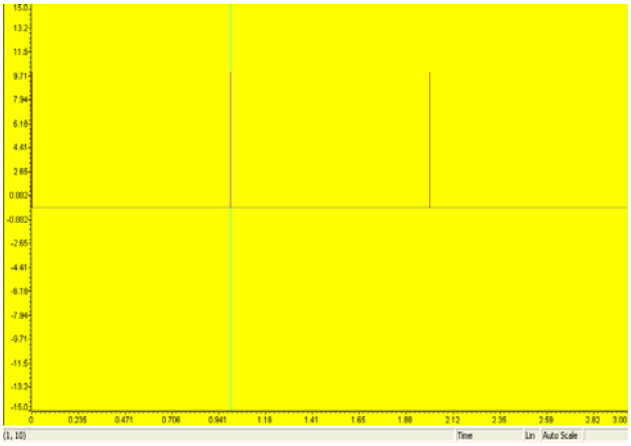
The corresponding output will be shown on the output window as shown below



11. To see the Graph go to View and select time/frequency in the Graph, and give the correct Start address provided in the program, Display data can be taken as per user.



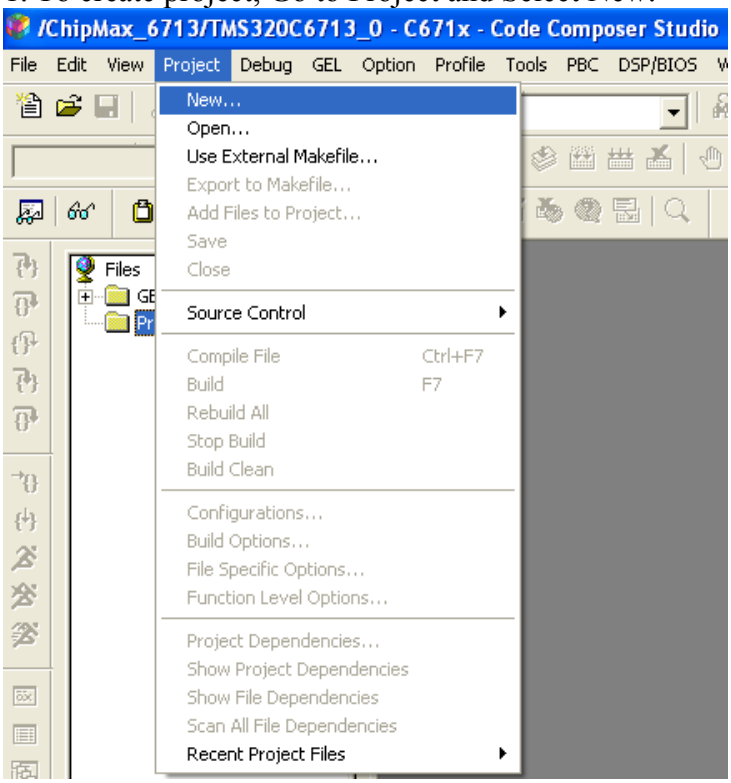




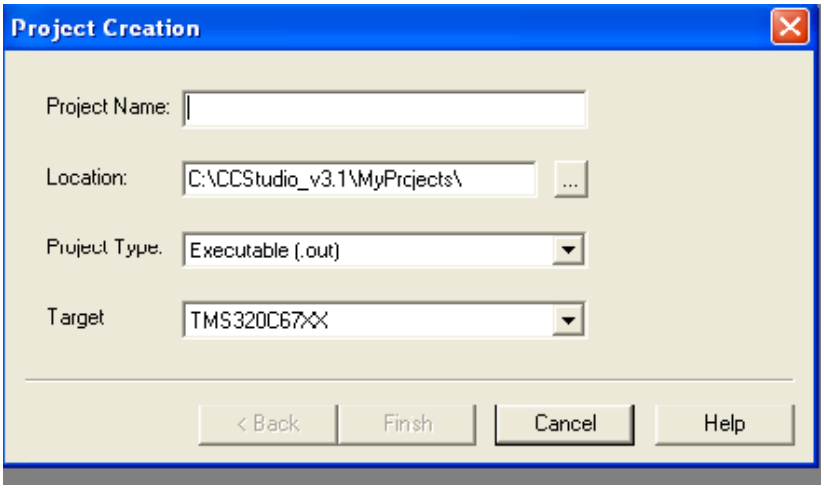
Experiment 3: N-Point DFT

Procedure to create new Project:

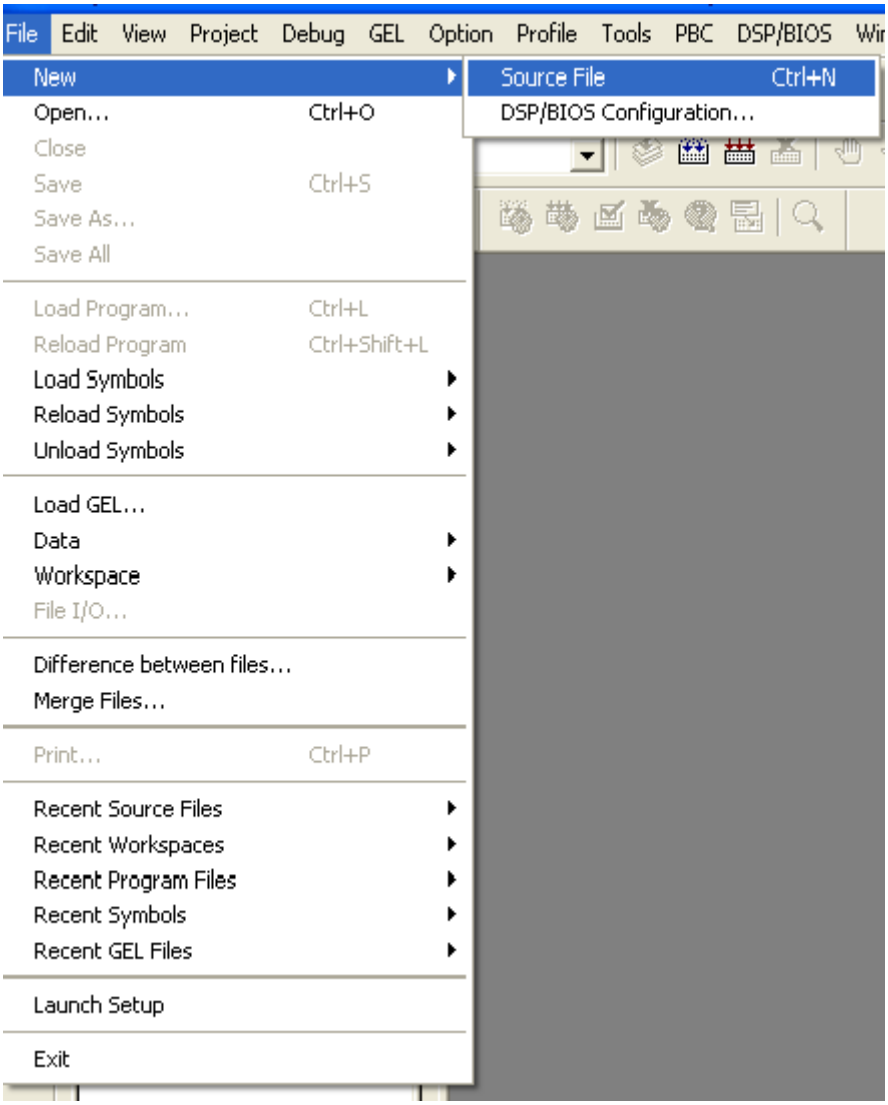
1. To create project, Go to Project and Select New.



2. Give project name and click on finish.



(**Note:** Location must be c:\CCStudio_v3.3\MyProjects).
3. Click on File ⇨ New ⇨ Source File, To write the Source Code.



AIM: TO COMPUTE N-POINT DFT OF A GIVEN SEQUENCE AND TO PLOT MAGNITUDE AND PHASE SPECTRUM.

Discrete Fourier Transform: The Discrete Fourier Transform is a powerful computation tool which allows us to evaluate the Fourier Transform $X(e^{j\omega})$ on a digital computer or specially designed digital hardware. Since $X(e^{j\omega})$ is continuous and periodic, the DFT is obtained by sampling one period of the Fourier Transform at a finite number of frequency points. Apart from determining the frequency content of a signal, DFT is used to perform linear filtering operations in the frequency domain.

The sequence of N complex numbers $x_0,..., x_{N-1}$ is transformed into the sequence of N complex numbers $X_0, ..., X_{N-1}$ by the DFT according to the formula:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \qquad k = 0,1, \ldots. N-1$$

Program:

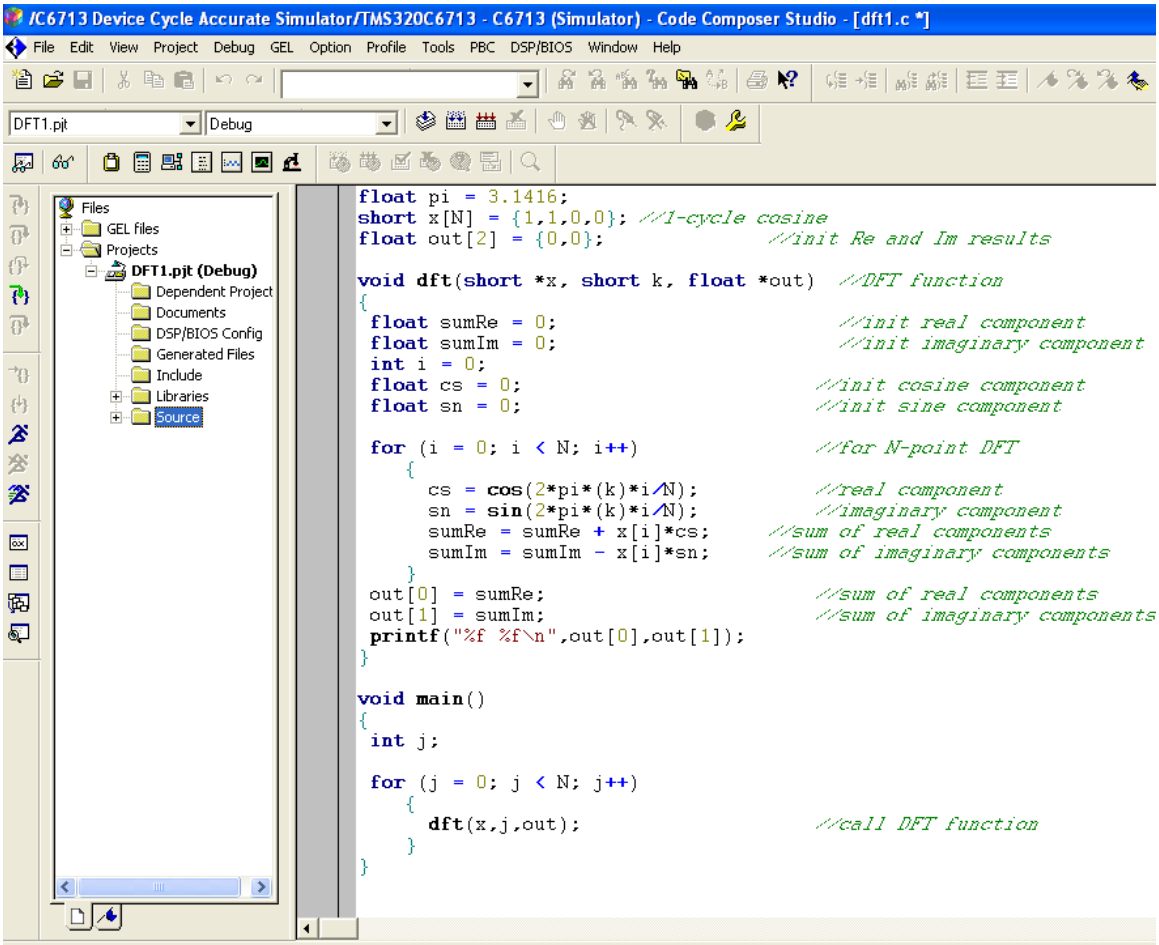
```
#include <stdio.h>
#include <math.h>
#define N 4                                          //number of data values

float pi = 3.1416;
short x[N] = {1,1,0,0};
float real[10],imaginary[10];                        //Output results
int index=0;

void dft(short *x, short k, float *out)  //DFT function
{
 float sumRe = 0;                  //initialize real component
 float sumIm = 0;                  //initialize imaginary component
 int i = 0;
 float cs = 0;                    //initialize cosine component
 float sn = 0;                   //initialize sine component

 for (i = 0; i < N; i++)              //for N-point DFT
   {
    cs = cos(2*pi*(k)*i/N);          //real component
    sn = sin(2*pi*(k)*i/N);          //imaginary component
    sumRe = sumRe + x[i]*cs;                //sum of real components
    sumIm = sumIm - x[i]*sn;                //sum of imaginary components
   }
 real[index]=sumRe;                     //sum of real components
 imaginary[index]=sumIm;               //sum of imaginary components
 index++;
 printf("%f %f\n",sumRe,sumIm);
```

```
}

void main()
{
 int j;
 for (j = 0; j < N; j++)
    {
      dft(x,j,out);                    //call DFT function
    }
}
```

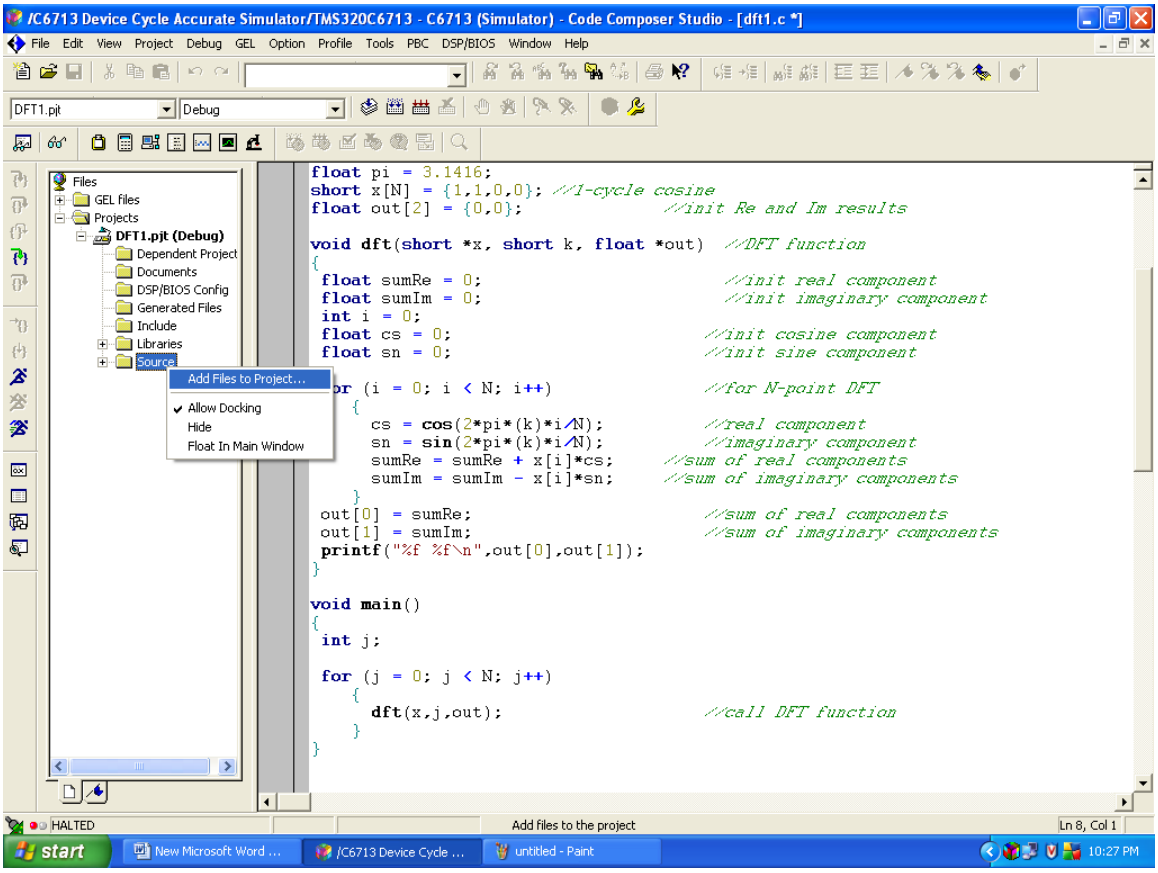
Output:

```
2.000000 0.000000
0.999996 -1.000000
0.000000 0.000007
1.000011 1.000000
```

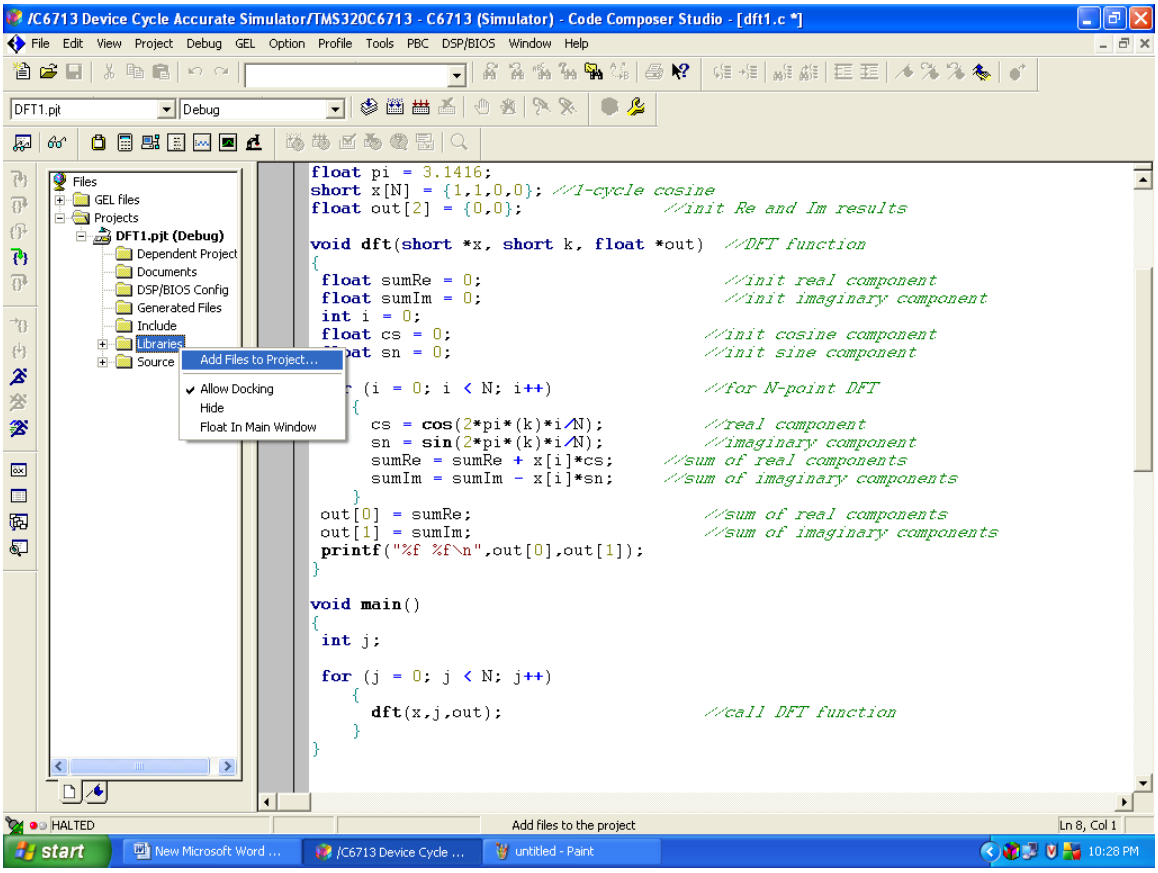
4. Enter the source code and save the file with "**.C**" extension.

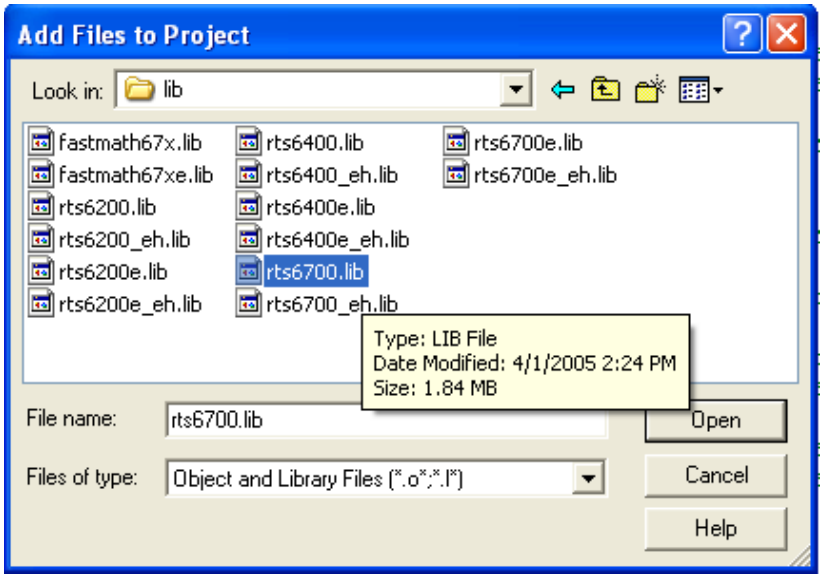


5. Right click on source, Select add files to project .. and Choose "**.C** " file Saved before.

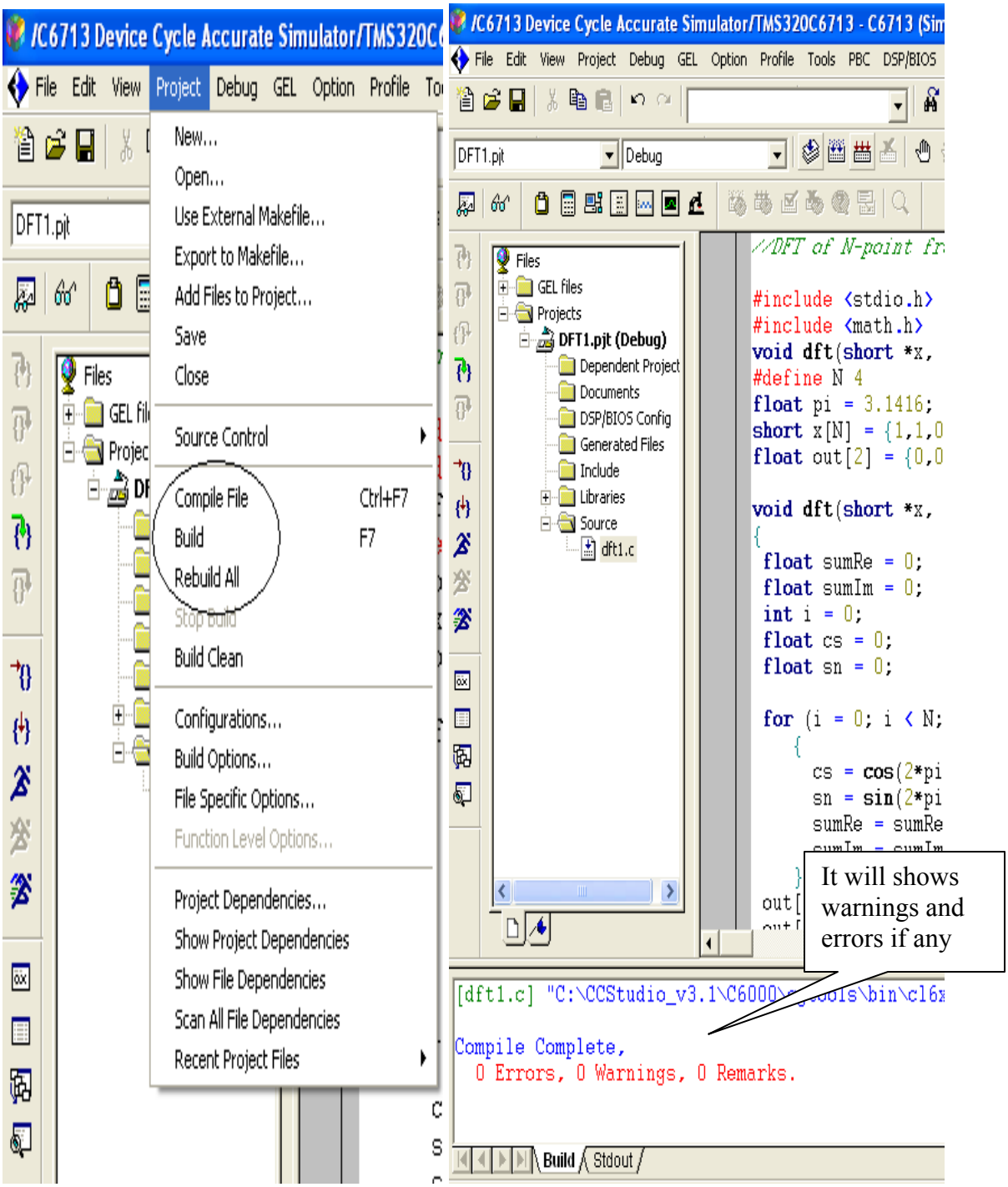


6. Right Click on libraries and select add files to project and choose C:\CCStudio_v3.3\C6000\cgtools\lib\rts6700.lib and click open.

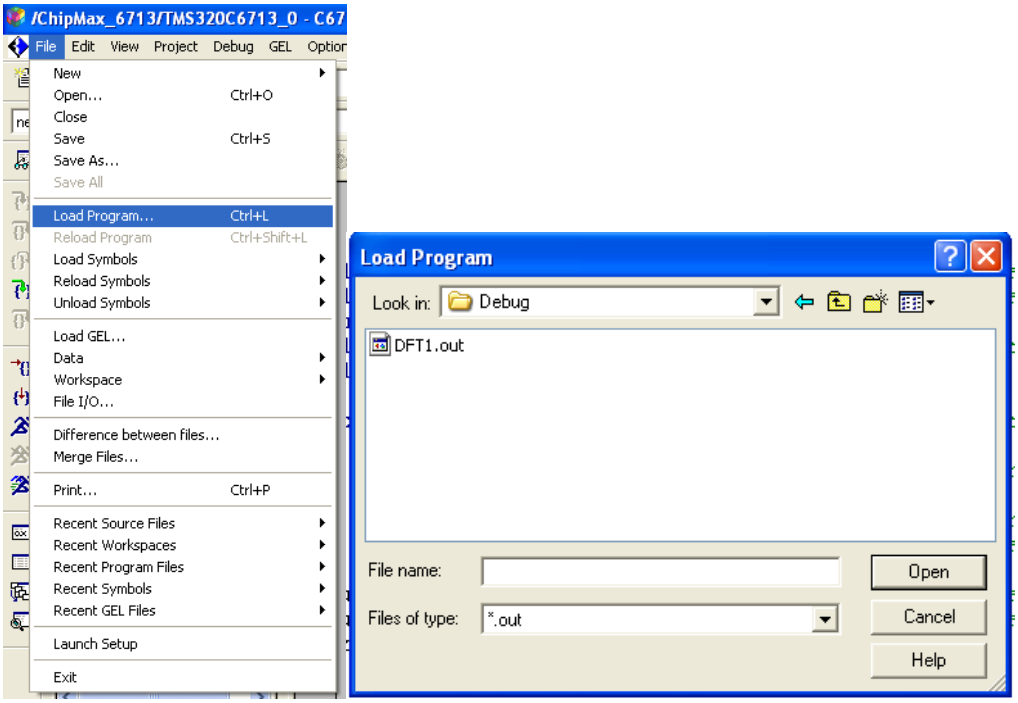




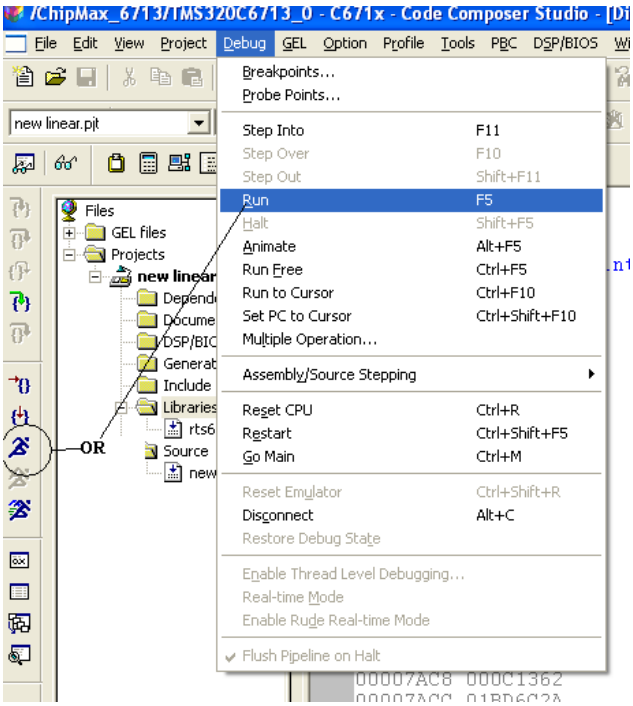
7. a) Go to Project to Compile .
   b) Go to Project to Build.
   c) Go to Project to Rebuild All.



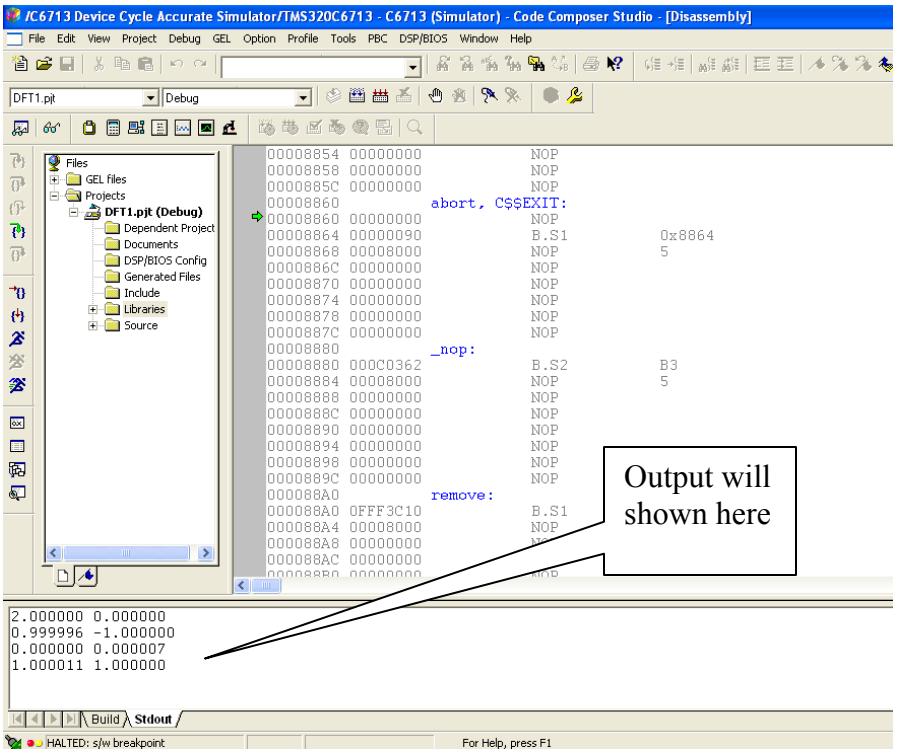
8. Go to file and load program and load **“.out”** file into the board..



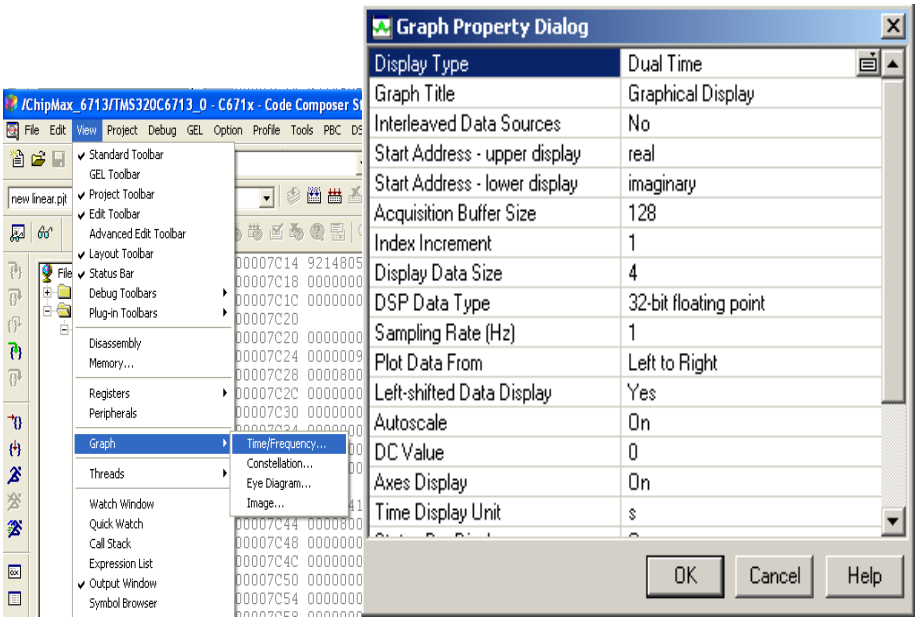
9. Go to Debug and click on run to run the program.



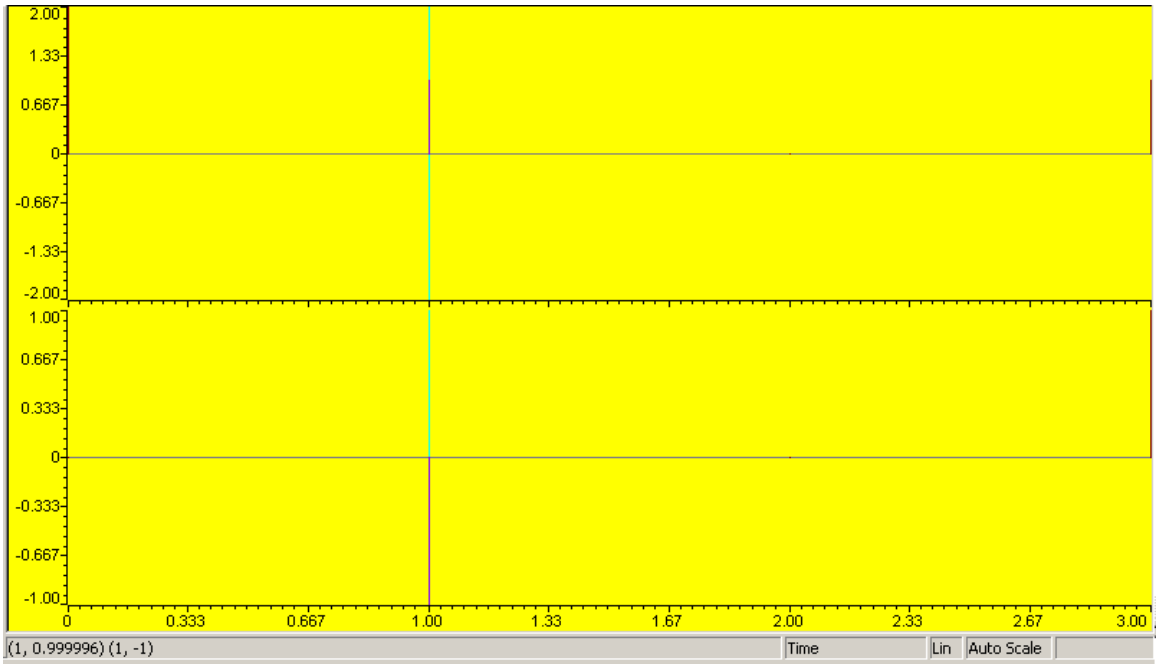
10. Observe the output in output window.



11. To see the Graph go to View and select time/frequency in the Graph, and give the correct Start address provided in the program, Display data can be taken as per user.



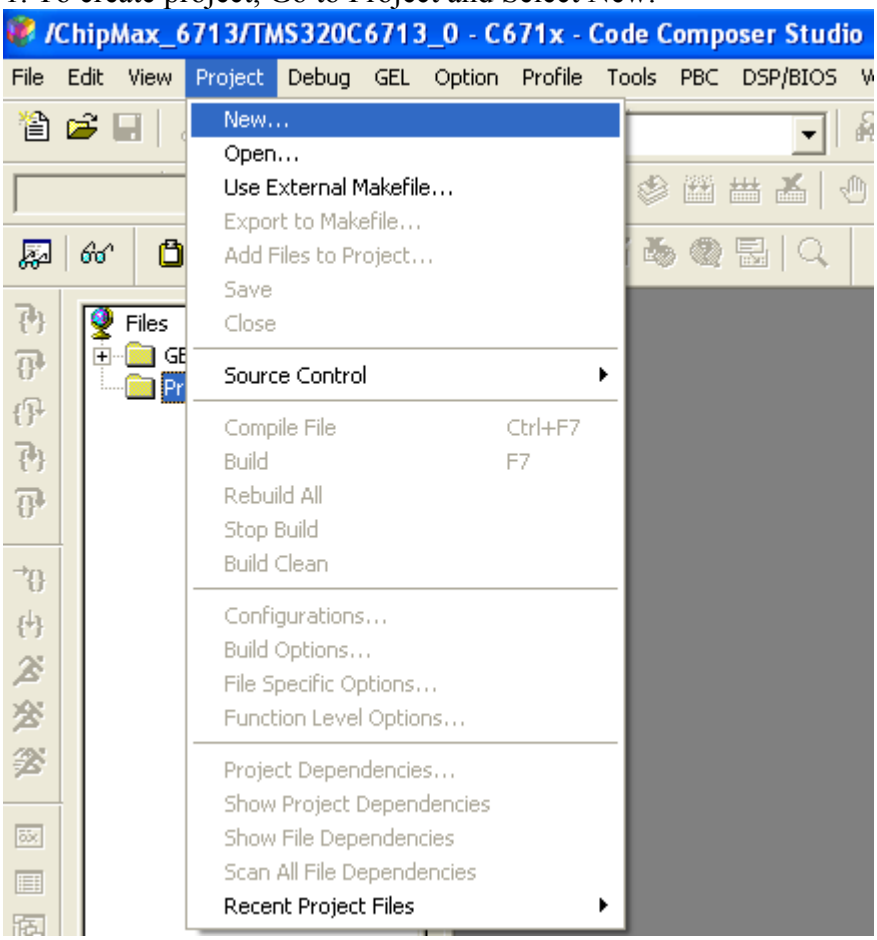
12. Green line is to choose the point, Value at the point can be seen (Highlighted by circle at the left corner).



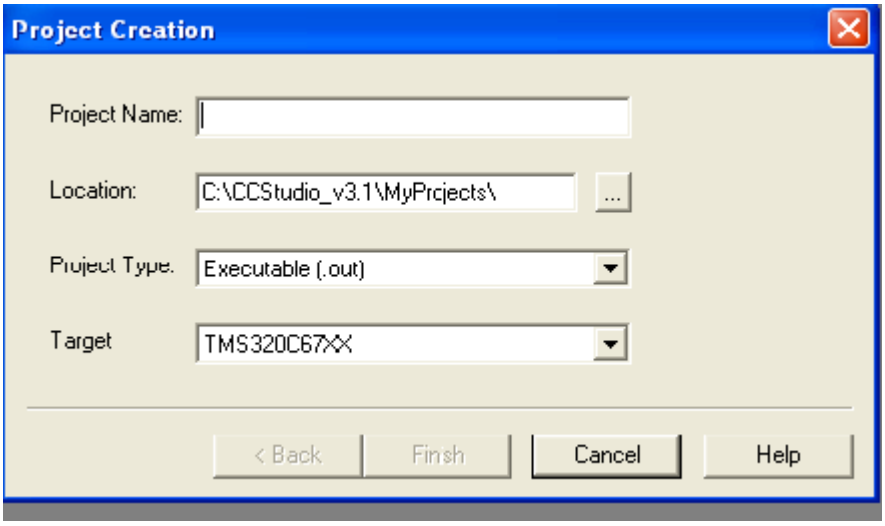
Experiment 4: FIR FILTER

Procedure to create new Project:

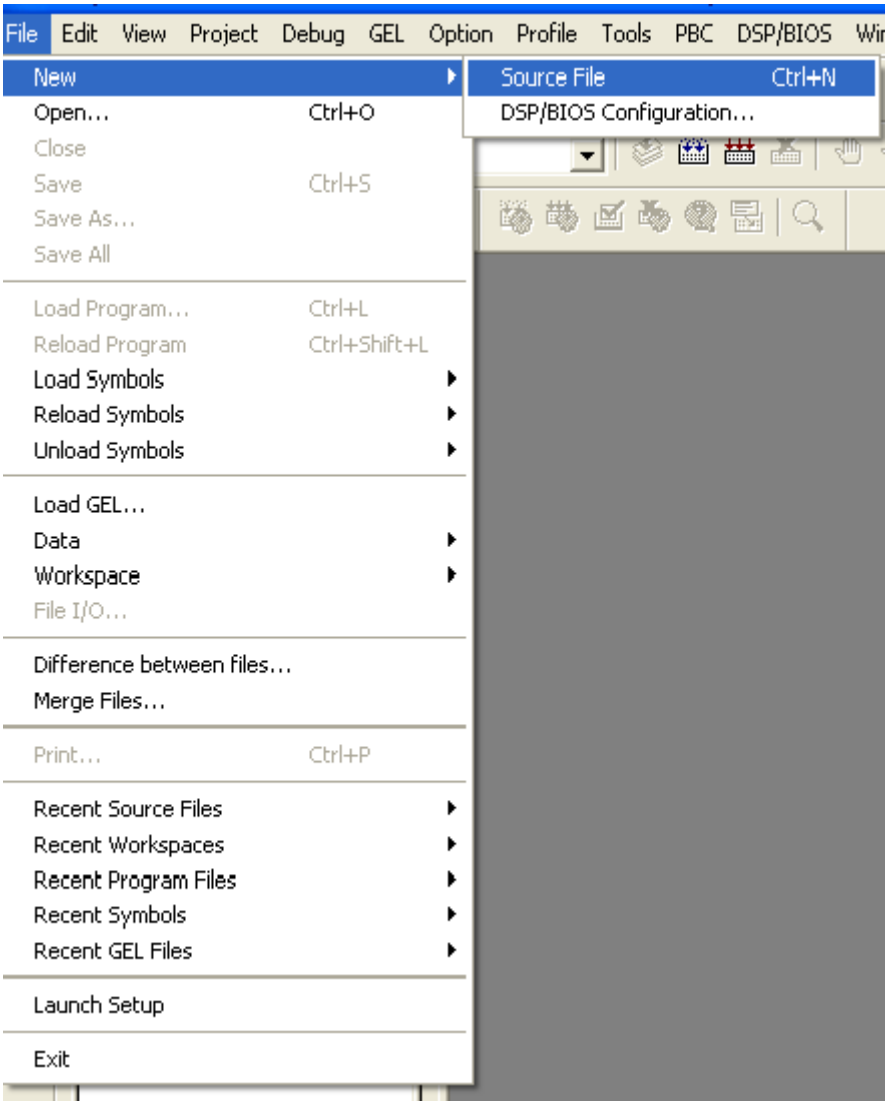
1. To create project, Go to Project and Select New.



2. Give project name and click on finish.



(Note**:** Location must be c:\CCStudio_v3.3\MyProjects).
3. Click on File ⇨ New ⇨ Source File, to write the Source Code.



AIM: Realization of FIR filter (any type) to meet given specifications. The input can be a signal from Function Generator/Speech signal

Finite Impulse Response (FIR) Filter: The FIR filters are of non-recursive type, whereby the present output sample is depending on the present input sample and previous input samples.

The transfer function of a FIR causal filter is given by,

$$H(z) = \sum_{n=0}^{N-1} h(n)z^{-n}$$

Where $h(n)$ is the impulse response of the filter.

The Fourier transform of $h(n)$ is

$$H(e^{jw}) = \sum_{n=0}^{N-1} h(n)e^{-jwn}$$

In the design of FIR filters most commonly used approach is using windows.

The desired frequency response $H_d(e^{jw})$ of a filter is periodic in frequency and can be expanded in Fourier series. The resultant series is given by,

$$h_d(n) = (1/2\pi) \int_{-\pi}^{\pi} H(e^{jw})e^{jwn}\, dw$$

And known as Fourier coefficients having infinite length. One possible way of obtaining FIR filter is to truncate the infinite Fourier series at $n = \pm [(N-1)/2]$

Where N is the length of the desired sequence.

The Fourier coefficients of the filter are modified by multiplying the infinite impulse response with a finite weighing sequence $w(n)$ called a window.

$$\text{Where } w(n) = w(-n) \neq 0 \quad \text{for } |n| \leq [(N-1)/2]$$
$$= 0 \quad \text{for } |n| > [(N-1)/2]$$

After multiplying $w(n)$ with $h_d(n)$, we get a finite duration sequence $h(n)$ that satisfies the desired magnitude response,

$$h(n) = h_d(n)\, w(n) \quad \text{for} \quad |n| \leq [(N-1)/2]$$
$$= 0 \quad \text{for} \quad |n| > [(N-1)/2]$$

The frequency response $H(e^{jw})$ of the filter can be obtained by convolution of $H_d(e^{jw})$ and $W(e^{jw})$ is given by,

$$H(e^{jw}) = (1/2\pi) \int_{-\pi}^{\pi} H_d(e^{j\theta})\, W(e^{j(w-\theta)})\, d\theta$$

$$H(e^{jw}) = H_d(e^{jw}) * W(e^{jw})$$

Program:

```
#include <stdio.h>
#include "c6713dsk.h"
#include "master.h"
#include "aic23cfg.h"
#include "dsk6713_aic23.h"
#define FilLen 32

// Low Pass Filter
/*float FilCo[32]= {   -0.0006,-0.0056,-0.0129,-0.0193,-0.0176,-0.0043,  0.0156,  0.0282,
0.0190,-0.0129,-0.0483,-0.0545,-0.0065,   0.0927,   0.2064,   0.2822,   0.2822,   0.2064,
0.0927,-0.0065,-0.0545,-0.0483,-0.0129,   0.0190,   0.0282,   0.0156,-0.0043,-0.0176,
0.0193,-0.0129,-0.0056,-0.0006};*/
```

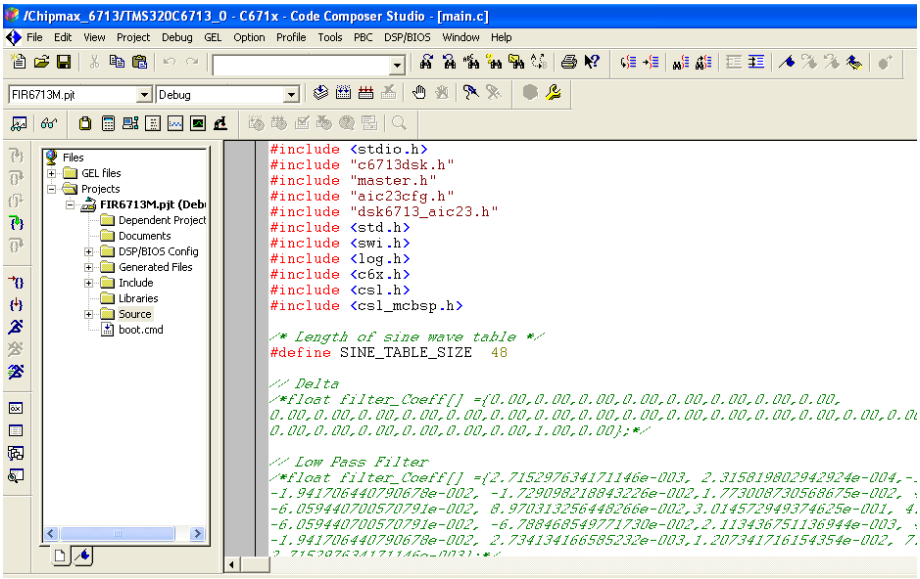
Note: //Generate the Coefficient using matlab filter design tool box //

```
// High Pass Filter
float  FilCo[32]  =  {  -0.0128,-0.0007,0.0068,0.0150,  0.0173,  0.0091,-0.0082,-0.0260,-
0.0317,-0.0160,0.0200,  0.0611,0.0808,0.0481,-0.0803,-0.5875,  0.5875,  0.0803,-0.0481,-
0.0808,-0.0611,-0.0200,0.0160,0.0317,  0.0260,  0.0082,-0.0091,-0.0173,-0.0150,-0.0068,
0.0007, 0.0128};

DSK6713_AIC23_Config config = {\
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
   0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
   0x00ff,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
   0x00ff,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */ \
   0x0010,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */      \
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */     \
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */             \
   0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
   0x008c, /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */            \
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};

DSK6713_AIC23_CodecHandle hCodec;
Int32 LInc = 4;
Int32 RInc = 3;
Int32 LData;
Int32 RData;
Int32 InChannel =0;
Int32 OtChannel =0;
Int32 LIndex = 0;
Int32 RIndex = 0;
Int32 LDataArr[128];
Int32 RDataArr[128];
Int32 LWP=0;
Int32 LRP=100;
```

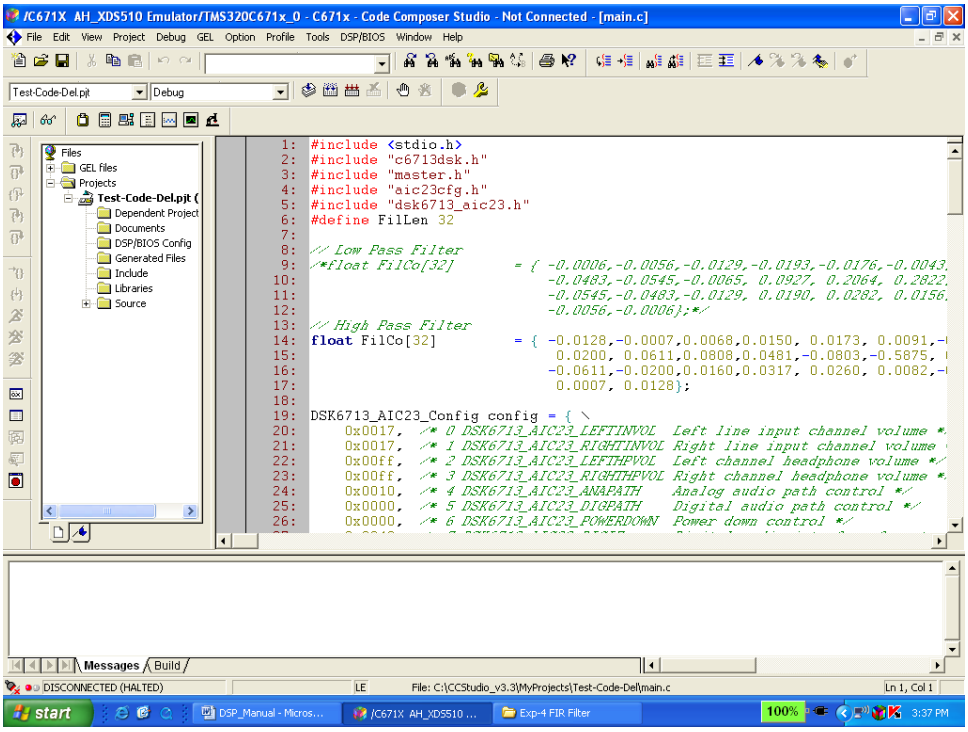
```
Int32 RWP=0;
Int32 RRP=100;
float LPrBuf[150];
float RPrBuf[150];
float FilOut= 0;

main(){
        int i;
        LED=0x0;
        CSL_init();
        // Filter Initialization
        for( i = 0 ; i < 100; i++ ) LPrBuf[i]=0.0;
        for( i = 0 ; i < 100; i++ ) RPrBuf[i]=0.0;

        // Initialize codec
        hCodec = DSK6713_AIC23_openCodec(0, &config);
        IRQ_globalEnable();
        IRQ_enable(IRQ_EVT_RINT1);
        IRQ_enable(IRQ_EVT_XINT1);
}
void led(){
        static int cc = 1;
        LED    = cc;
        // To Shift Pattern
        if (cc == 0x03) cc = 0x05;
        else if (cc == 0x05) cc = 0x06;
        else if (cc == 0x06) cc = 0x03;
        else cc = 0x03;
}
setpll200M(){

}
void read(){
        int i = 0;

        if (InChannel ==0){
                InChannel =1;
                // Process Left Channel
                LData=MCBSP_read(DSK6713_AIC23_DATAHANDLE);
                for( i = 0; i <= FilLen-2; i++) LPrBuf[i] = LPrBuf[i+1];
                LPrBuf[FilLen-1] = (float) LData;
                FilOut = 0.0;
                for( i = 0 ; i <= FilLen-1; i++ )        FilOut += (FilCo[i] * LPrBuf[i]);
                LData = (int) (FilOut/16);
                LDataArr[LWP++] =  LData;
                if (LWP >= 128) LWP =0;
```

```
        }
        else{
                InChannel =0;
                // Process Right Channel
                RData=MCBSP_read(DSK6713_AIC23_DATAHANDLE);
                RDataArr[RWP++] = RData;
                if (RWP >= 128) RWP =0;
        }
}

void write(){
        if (OtChannel == 1){
                MCBSP_write(DSK6713_AIC23_DATAHANDLE,LDataArr[LRP++]);
                if (LRP >= 128)         LRP = 0;
                OtChannel = 0;
        }
        else{
                MCBSP_write(DSK6713_AIC23_DATAHANDLE,RDataArr[RRP++]);
                if (RRP >= 128)         RRP = 0;
                OtChannel = 1;
        }
}
```

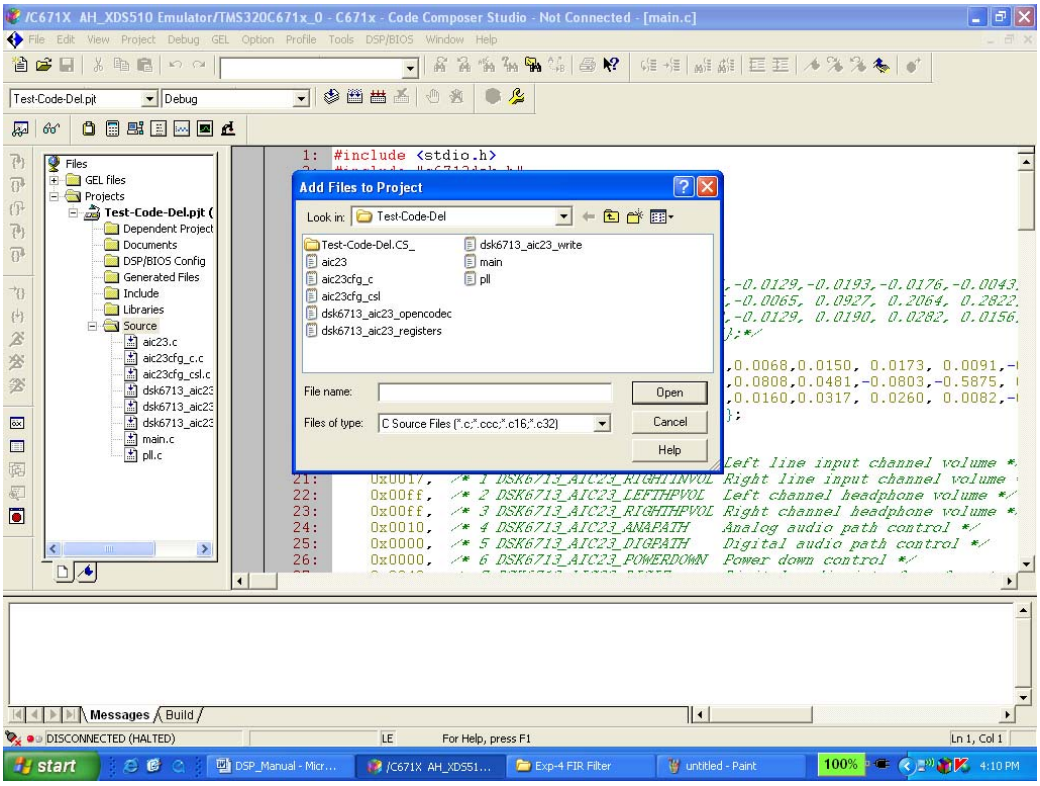
4. Enter the source code and save the file with main.c extension.



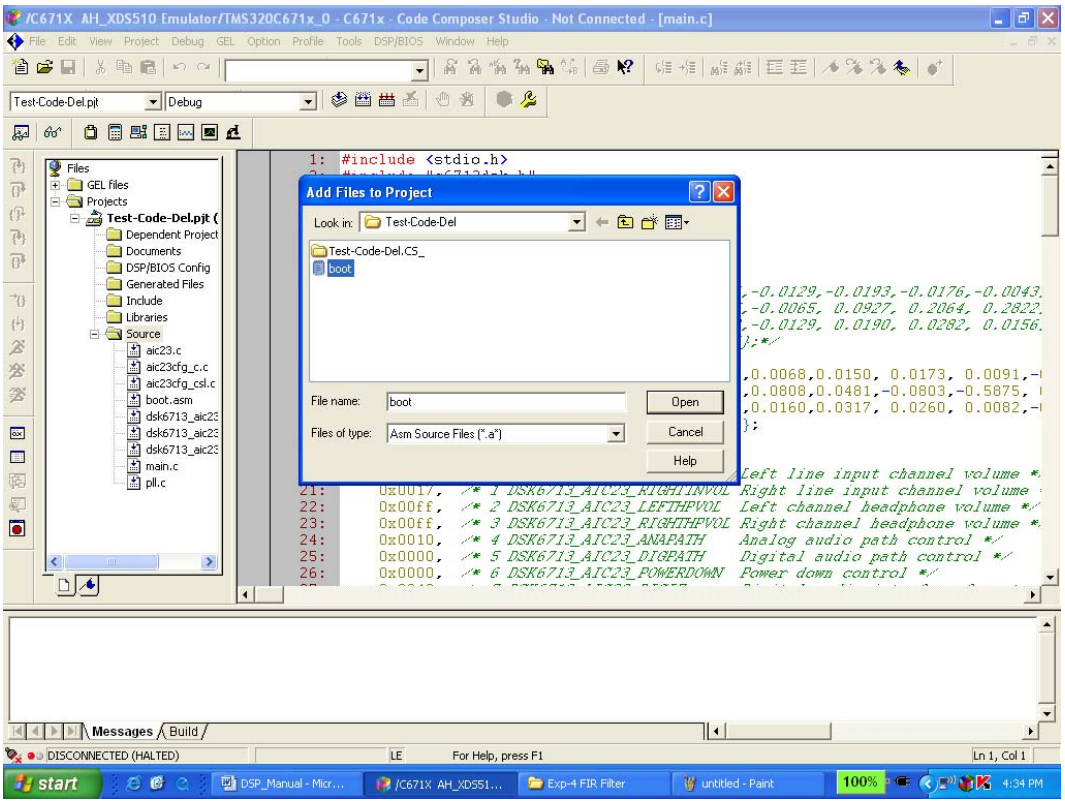
5. Right click on source, Select add files to project and Choose main.c file Saved before.



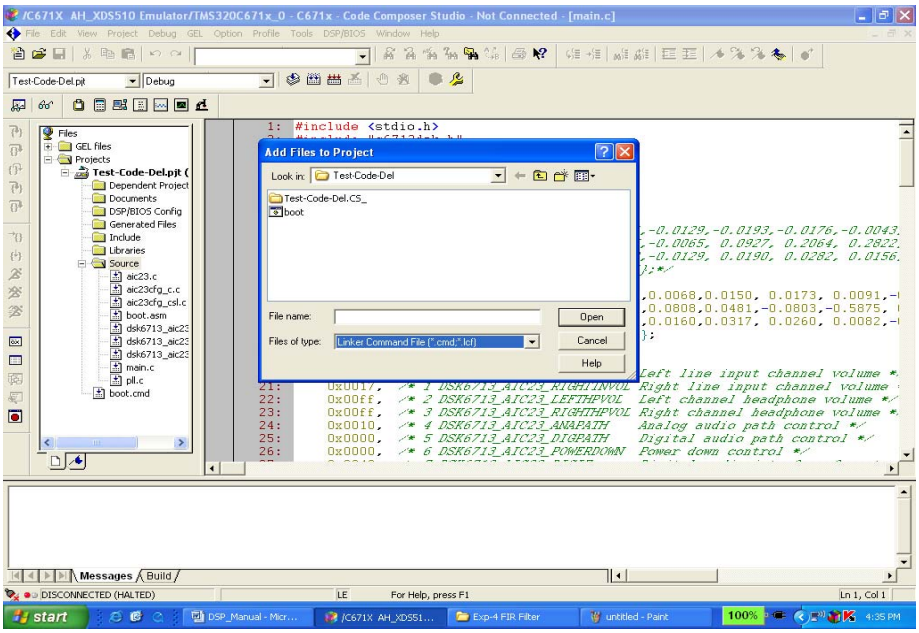
7. Right Click on the source add other supporting .c files which configure the audio codec.



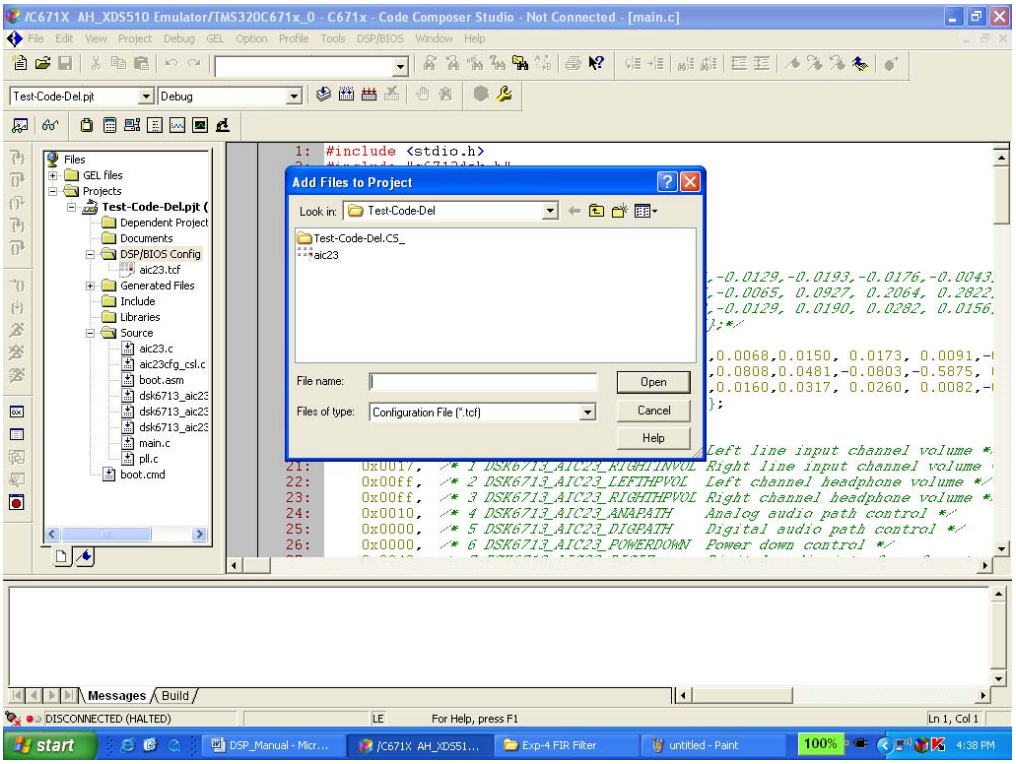
8. Right Click on the source add other supporting .asm files which configure the audio codec. i.e. boot.asm



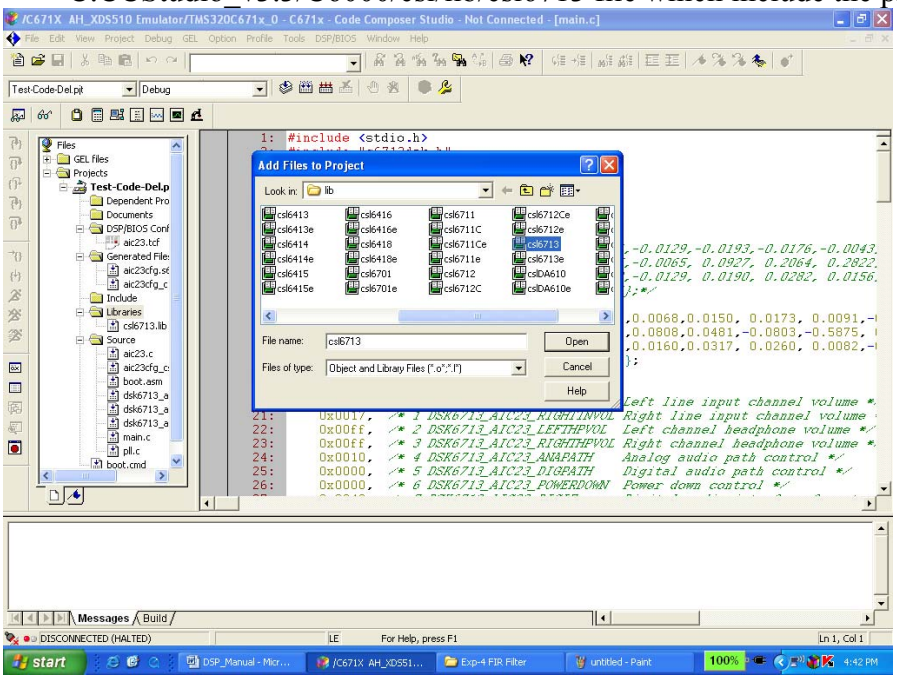
9. Right Click on the source add other supporting .asm files which configure the audio codec. i.e. boot.asm



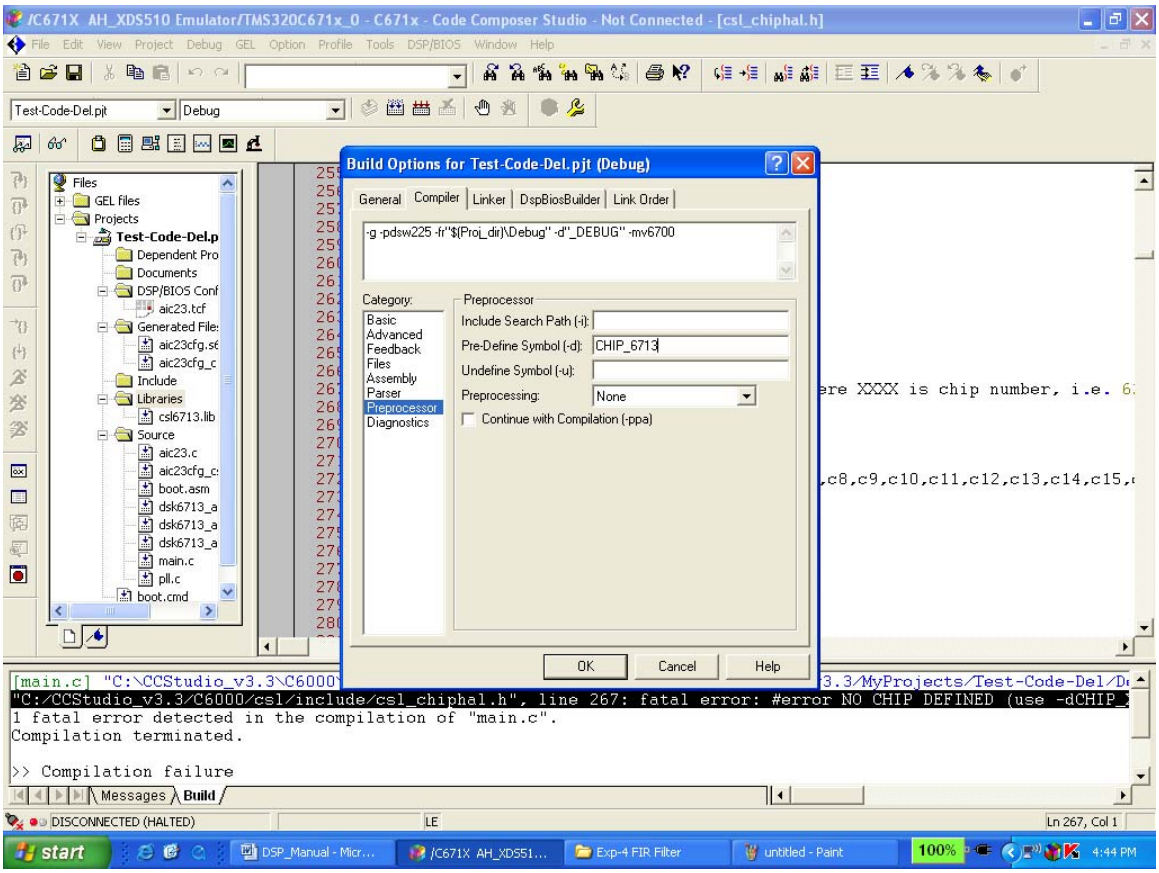
10. Right Click DSP/BIOS Config on the add aic23.tcf file which configure the DSP/BIOS.



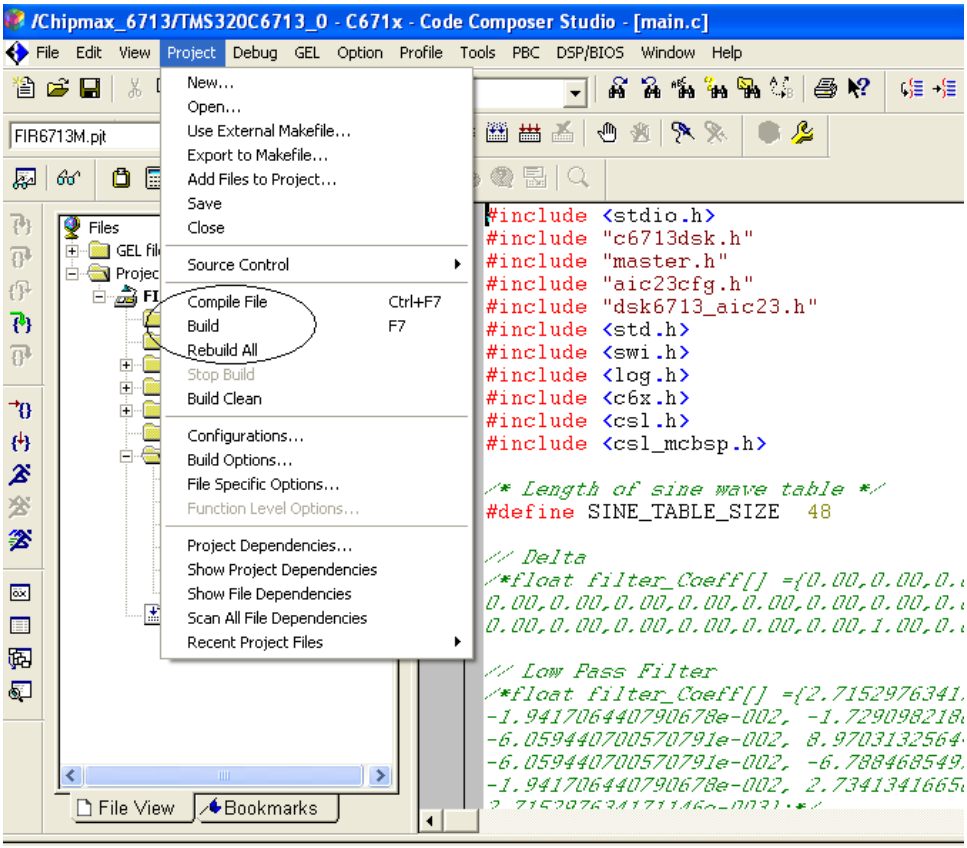
11. Right Click on Libraries add csl6713.lib from C:CCStudio_v3.3/C6000/csl/lib/csl6713 file which include the processor libraries.

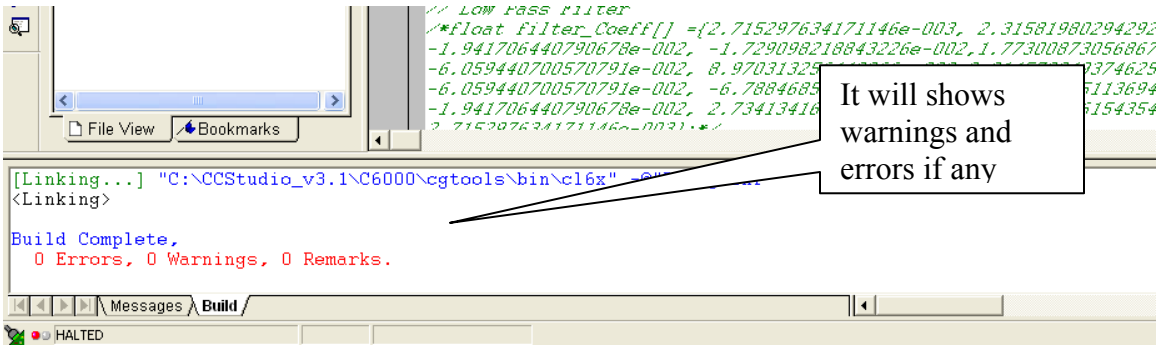


12 Go to Project-→ Build option → Preprocessor→set CHIP_6713 as shown below.

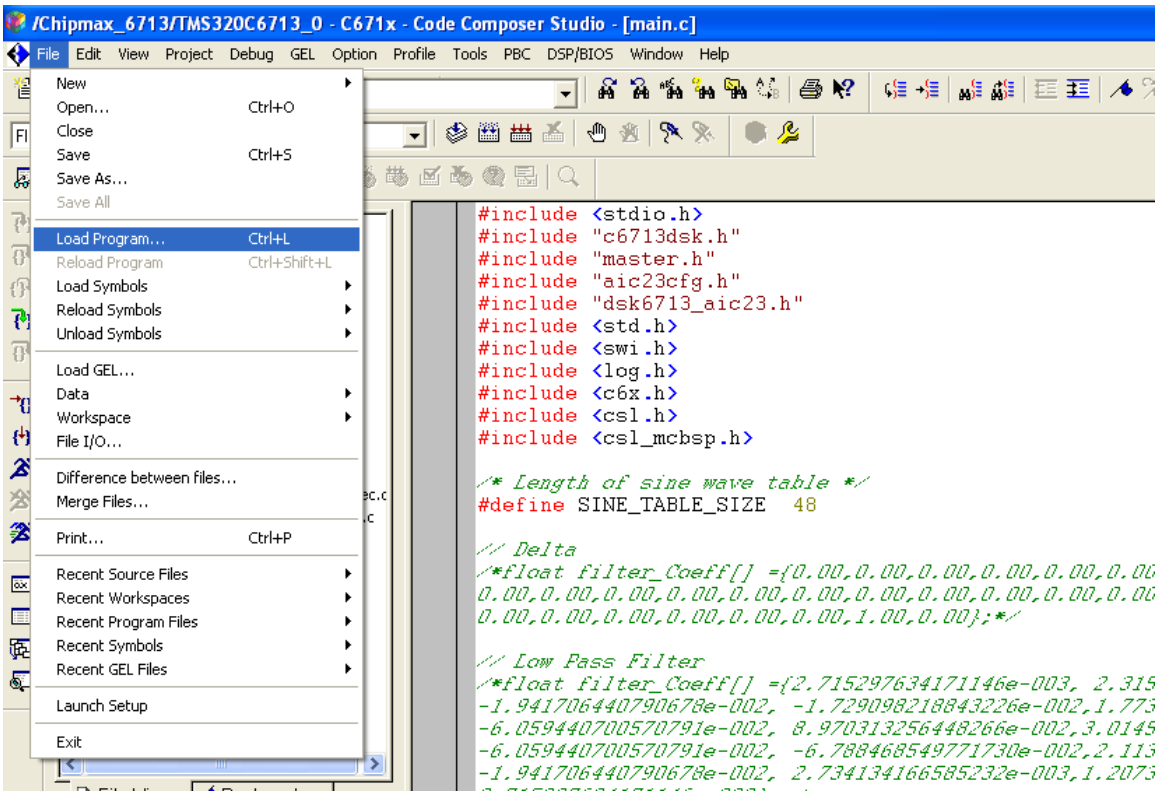


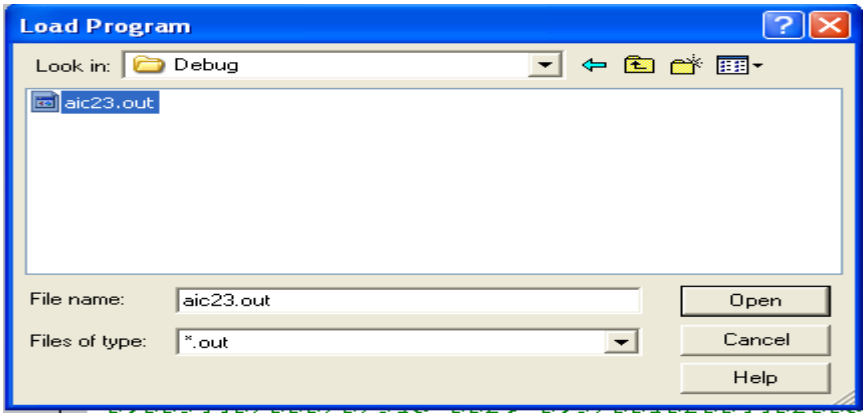
7. 7. a) Go to Project to Compile.
b) Go to Project to Build.
c) Go to Project to Rebuild All.


/Chipmax_6713/TMS320C6713_0 - C671x - Code Composer Studio - [main.c]
File Edit View Project Debug GEL Option Profile Tools PBC DSP/BIOS Window Help
New...
Open...
Use External Makefile...
Export to Makefile...
Add Files to Project...
Save
Close
Source Control
Compile File Ctrl+F7
Build F7
Rebuild All
Stop Build
Build Clean
Configurations...
Build Options...
File Specific Options...
Function Level Options...
Project Dependencies...
Show Project Dependencies
Show File Dependencies
Scan All File Dependencies
Recent Project Files
FIR6713M.pjt
Files
File View
Bookmarks
#include <stdio.h>
#include "c6713dsk.h"
#include "master.h"
#include "aic23cfg.h"
#include "dsk6713_aic23.h"
#include <std.h>
#include <swi.h>
#include <log.h>
#include <c6x.h>
#include <csl.h>
#include <csl_mcbsp.h>
/* Length of sine wave table */
#define SINE_TABLE_SIZE 48
// Delta
// Low Pass Filter

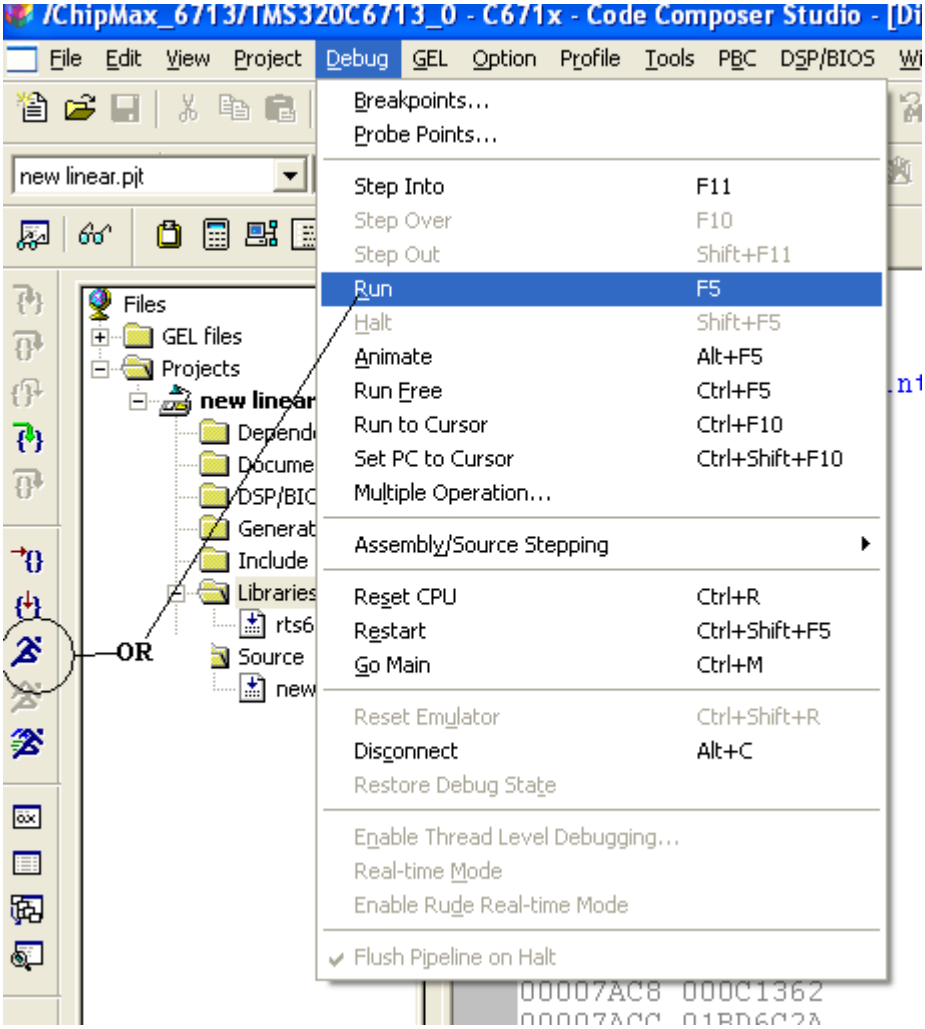

File View
Bookmarks
[Linking...] "C:\CCStudio_v3.1\C6000\cgtools\bin\cl6x"
<Linking>
Build Complete,
0 Errors, 0 Warnings, 0 Remarks.
Messages
Build
HALTED
It will shows warnings and errors if any

8. Go to file and load program and load **“.out”** file into the board.





9. Go to Debug and click on run to run the program.

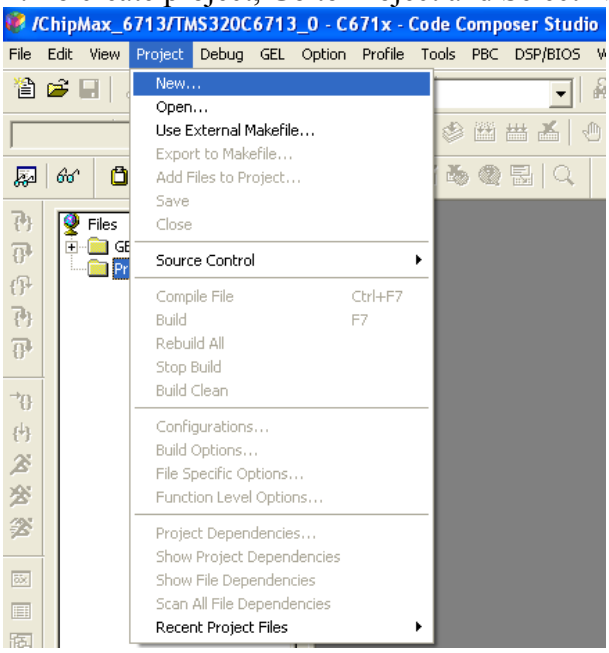


10. Output can be seen on CRO with the filtering effect.

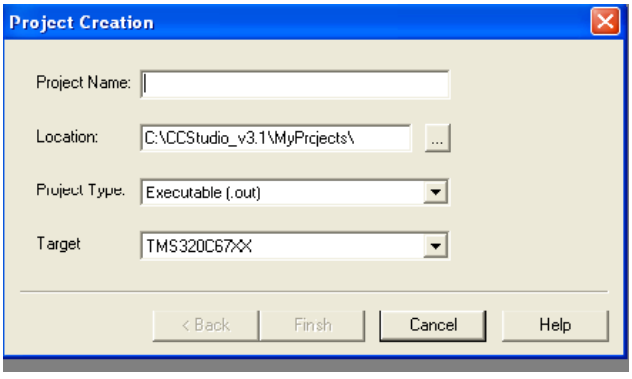
Experiment 5: To plot spectrum of cosine signal plus a signal from function generator

Procedure to create new Project:

1. To create project, Go to Project and Select New.

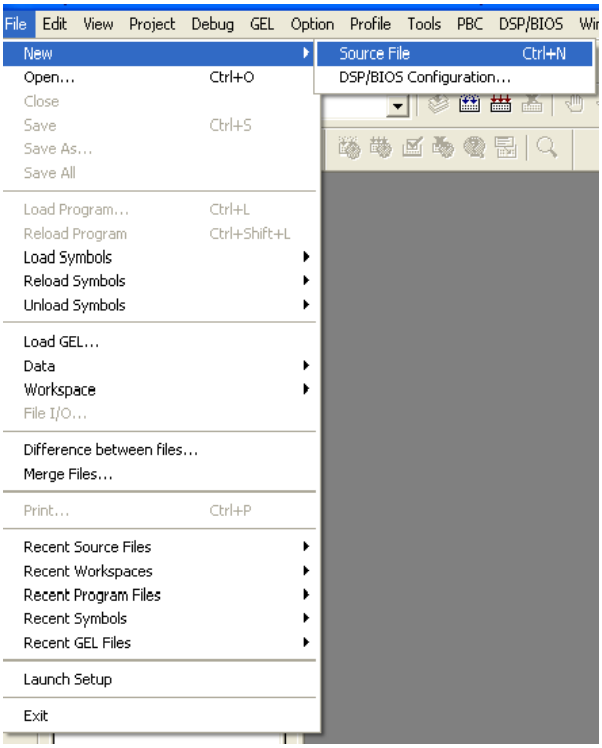


2. Give project name and click on finish.



(Note**:** Location must be c:\CCStudio_v3.3\MyProjects).

3. Click on File ⇨ New ⇨ Source File, to write the Source Code.



AIM: To add a cosine signal to a signal from a function generator/Microphone, and to plot the spectrum.

```
#include <stdio.h>
#include <csl.h>
#include "c6713dsk.h"
#include "master.h"
#include "aic23cfg.h"
#include "dsk6713_aic23.h"

/* Length of sine wave table */
#define SINE_TABLE_SIZE 48

// Low Pass Filter
float filter_Coeff[] ={2.715297634171146e-003, 2.315819802942924e-004,
1.244493581373643e-002, 7.244364917221401e-003,1.207341716154354e-002,
2.734134166585232e-003,-1.941706440790678e-002, -1.729098218843226e-
002,1.773008730568675e-002, 4.091495174059349e-002,2.113436751136944e-003, -
6.788468549771730e-002,-6.059440700570791e-002, 8.970313256448266e-
```

```
002,3.014572949374625e-001, 4.019009454299968e-001,3.014572949374625e-001,
8.970313256448266e-002,-6.059440700570791e-002, -6.788468549771730e-
002,2.113436751136944e-003, 4.091495174059349e-002,1.773008730568675e-002, -
1.729098218843226e-002,-1.941706440790678e-002,
2.734134166585232e003,1.207341716154354e-002, 7.244364917221401e-
003,1.244493581373643e-002, 2.315819802942924e-004,2.715297634171146e-003};

// High Pass Filter
/*float filter_Coeff[] ={3.294316420702696e-004,3.800020076486443e-
003,9.822200806739014e-003,1.517265313889167e-002, 1.323547007544908e-
002,2.635896986048919e-004,-1.808215737734512e-002,-2.666833013269758e-002, -
1.155354962270025e-002,2.448211866656400e-002,5.534101055783895e
002,4.424359087198896e-002, -2.922329551555757e-002,-1.473332022689261e-001,-
2.574625659073934e-001,6.976203109125493e-001, -2.574625659073934e-001,
1.473332022689261e-001,-2.922329551555757e-002,4.424359087198896e-002,
5.534101055783895e-002,2.448211866656400e-002,-1.155354962270025e-002,
2.666833013269758e-002, -1.808215737734512e-002,2.635896986048919e
004,1.323547007544908e-002,1.517265313889167e-002, 9.822200806739014e-
003,3.800020076486443e-003,3.294316420702696e-004};*/

// Pre-generated Co-sine wave data, 16-bit signed samples
Int32 sinetable[SINE_TABLE_SIZE] = { 0x00000000, 0x000010b4, 0x00002120,
0x000030fb, 0x00003fff, 0x00004dea, 0x00005a81, 0x0000658b,0x00006ed8,
0x0000763f, 0x00007ba1, 0x00007ee5, 0x00007ffd, 0x00007ee5, 0x00007ba1,
0x000076ef,  0x00006ed8, 0x0000658b, 0x00005a81, 0x00004dea, 0x00003fff,
0x000030fb, 0x00002120, 0x000010b4, 0x00000000, 0xffffef4c, 0xffffdee0, 0xffffcf06,
0xffffc002, 0xffffb216, 0xffffa57f, 0xffff9a75, 0xffff9128, 0xffff89c1, 0xffff845f,
0xffff811b, 0xffff8002, 0xffff811b, 0xffff845f, 0xffff89c1,0xffff9128, 0xffff9a76,
0xffffa57f, 0xffffb216, 0xffffc002, 0xffffcf06, 0xffffdee0, 0xffffef4c
};
```

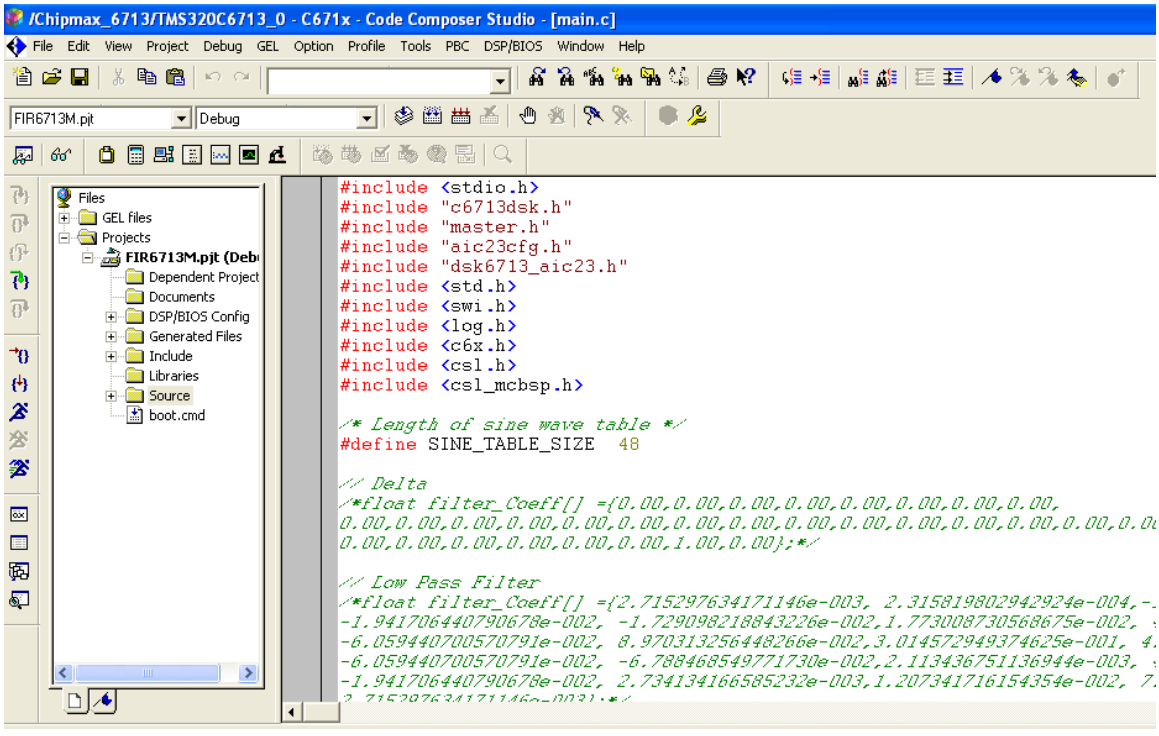
Note: //Generate sine table from matlab//

```
DSK6713_AIC23_Config config = { \
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
   0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
   0x00ff,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
         0x00ff,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume
*/ \
   0x0010,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */      \
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */     \
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */             \
   0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
         0x008c,          /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */
\
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};
```

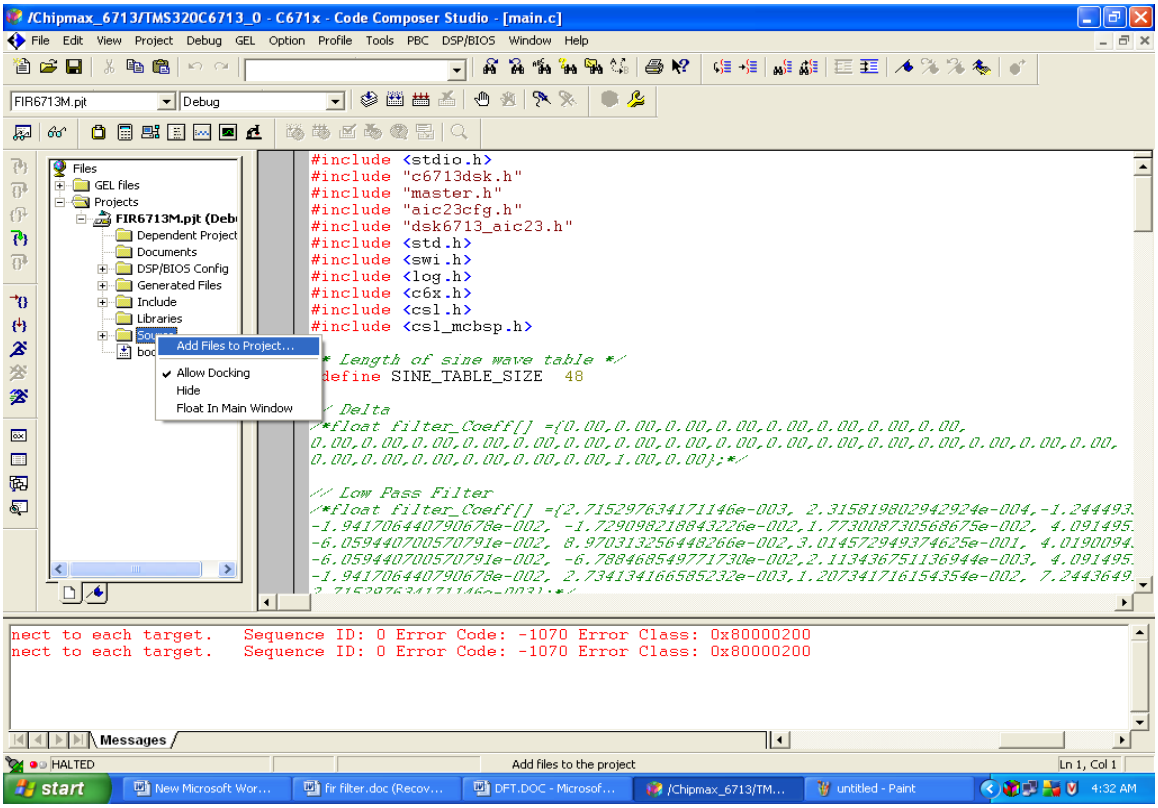
```
DSK6713_AIC23_CodecHandle hCodec;
Int32 InitWait =1;
Int32 data;
Int32 Logger[1024];
Int32 LoggerIndex =0;
Int32 Avl =0;
float in_buffer[100];


main()
{

        int i;
        LED=0x0;
        CSL_init();

        // Filter Initialization
        for( i = 0 ; i < 100; i++ ) in_buffer[i]=0.0;
        // Initialize codec
        hCodec = DSK6713_AIC23_openCodec(0, &config);
        IRQ_globalEnable();
        IRQ_enable(IRQ_EVT_RINT1);
        IRQ_enable(IRQ_EVT_XINT1);
}



void led(){
        static int cc = 1;
        LED    = cc;
        // To Shift Pattern
        if (cc == 0x03) cc = 0x05;
        else if (cc == 0x05) cc = 0x06;
        else if (cc == 0x06) cc = 0x03;
        else cc = 0x03;
}

setpll200M(){

}

void read(){
        //int i =0;
//      float result;
        static int Index =0;
```

```
        data=MCBSP_read(DSK6713_AIC23_DATAHANDLE);
        data += sinetable[Index];
        Index+=2;
        if (Index >=48) Index %=48;
        Avl = 1;
        Logger[LoggerIndex++] = data;
        if (LoggerIndex == 1024)
                LoggerIndex = 0;
}

void write(){
        if (Avl ==1){
                MCBSP_write(DSK6713_AIC23_DATAHANDLE, data);
                Avl =0;
        }
        else
                MCBSP_write(DSK6713_AIC23_DATAHANDLE, data);
}
```

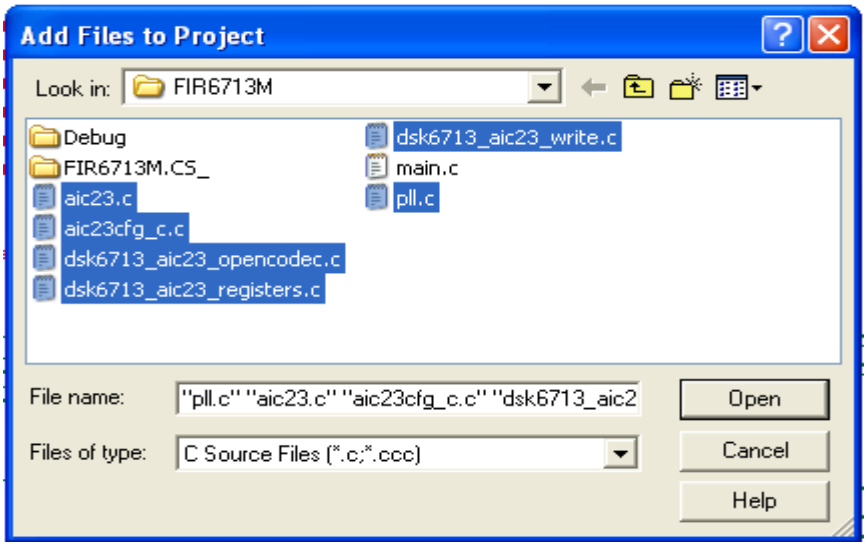
4. Enter the source code and save the file with main.c extension.



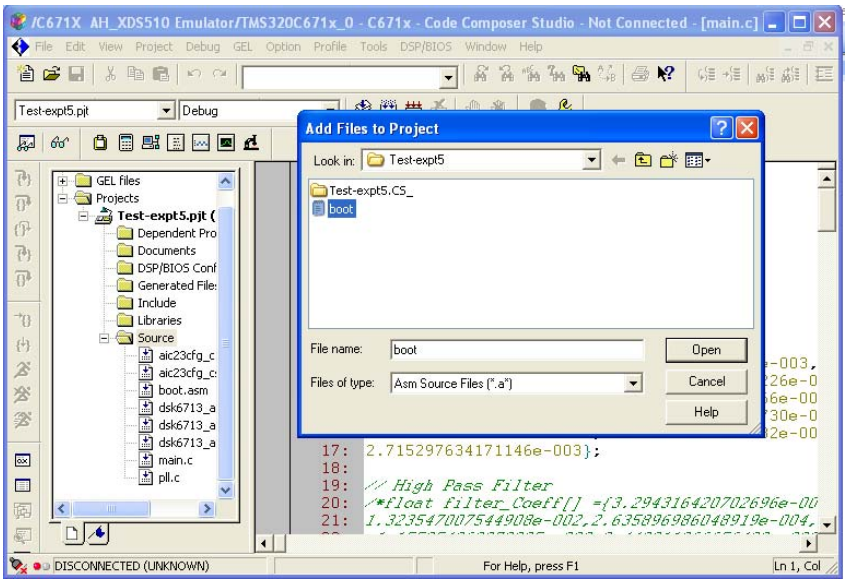
5. Right click on source, Select add files to project and Choose main.c file Saved before.



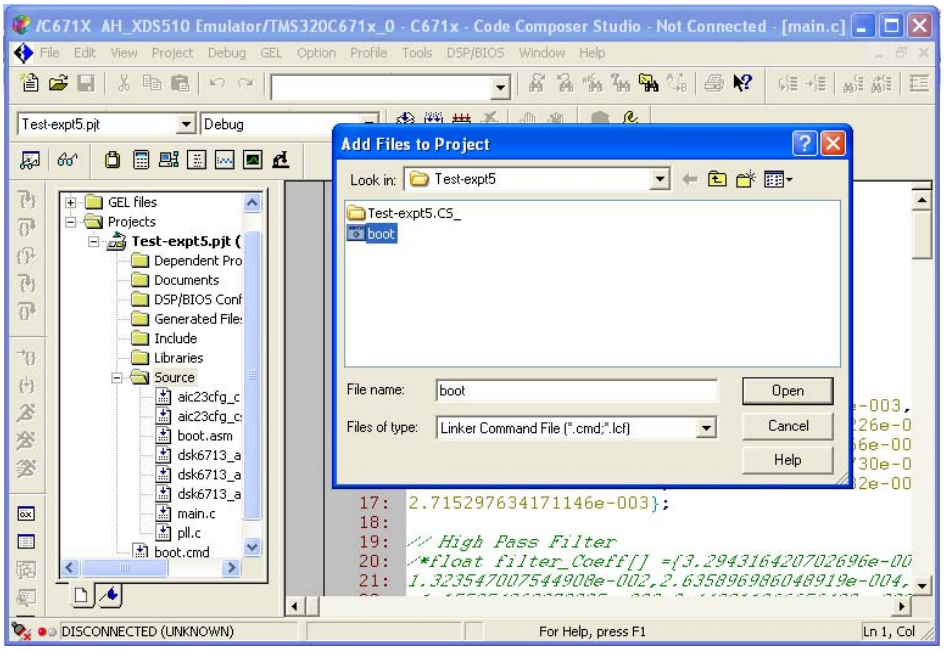
6. Add the other supporting .c files which configure the audio codec.



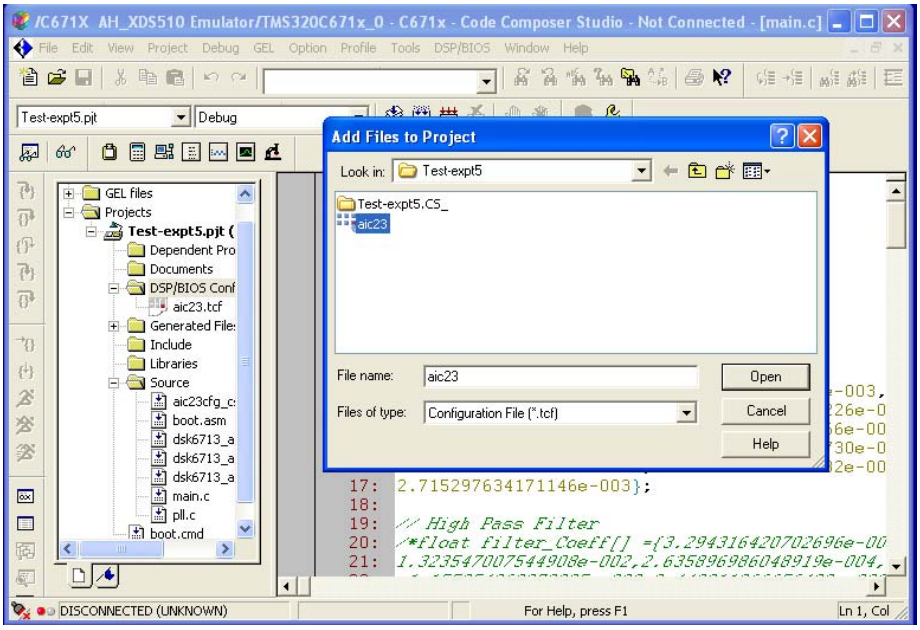
7 Right click on source, Select add files to project and Choose boot.asm and save it.



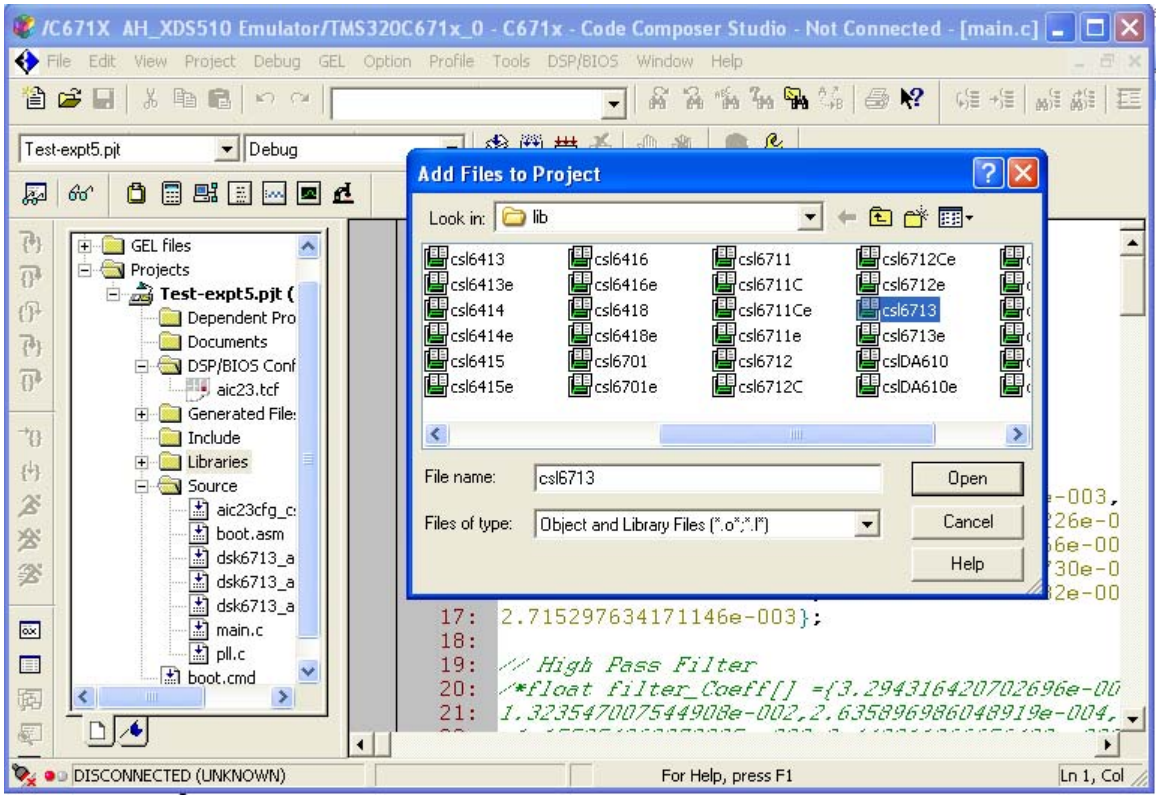
8 Right click on source, select add files to project and choose boot.cmd and save it.



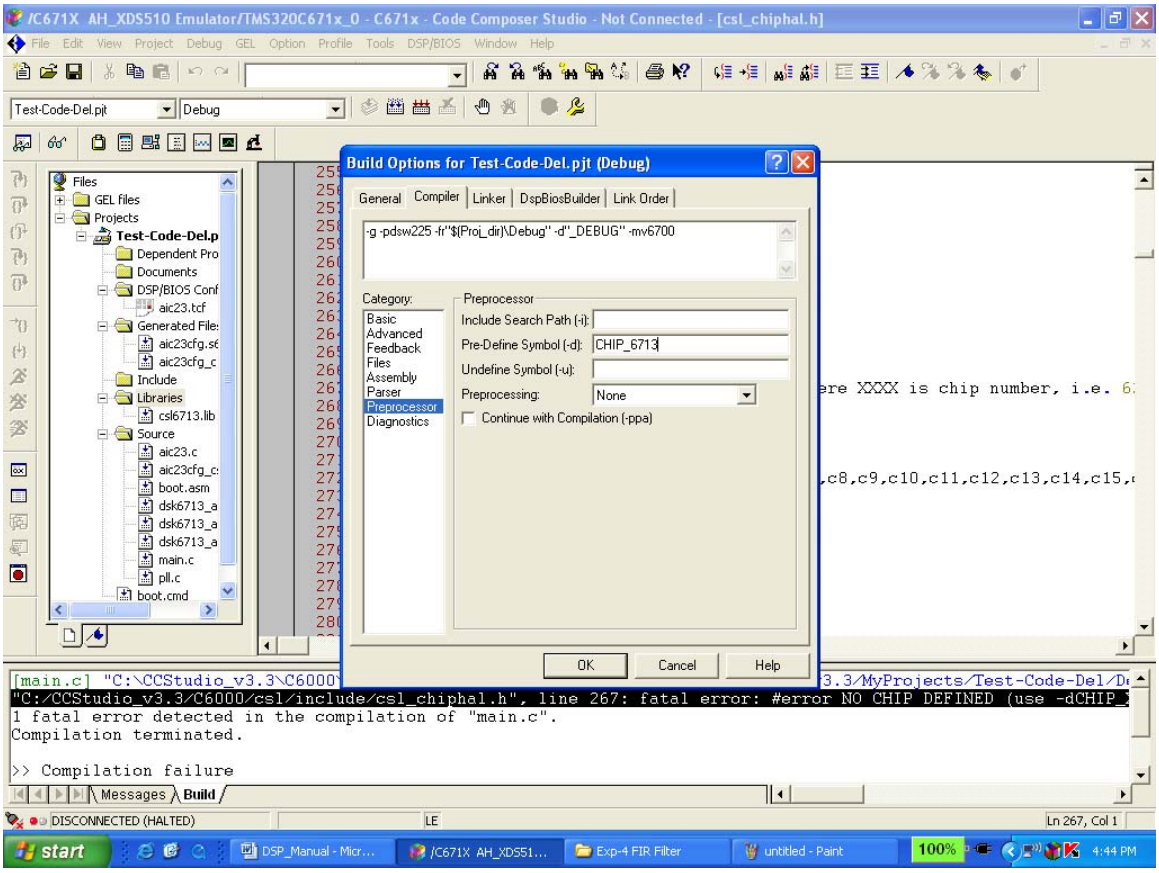
9 Right click on DSP/BIOS Config and select add files project and add aic23.tcf file and save it.



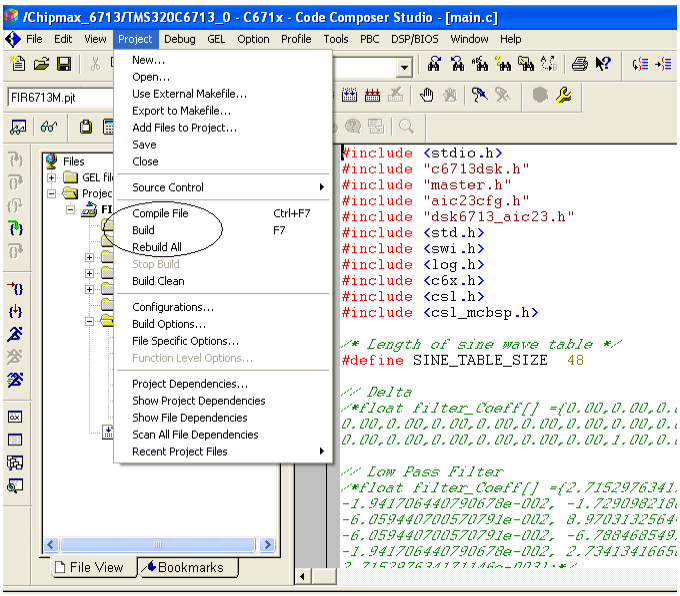
8 Right click on Libraries and select add files to project add csl6713.lib from C:CCStudio_v3.3/C6000/csl/lib/csl6713.

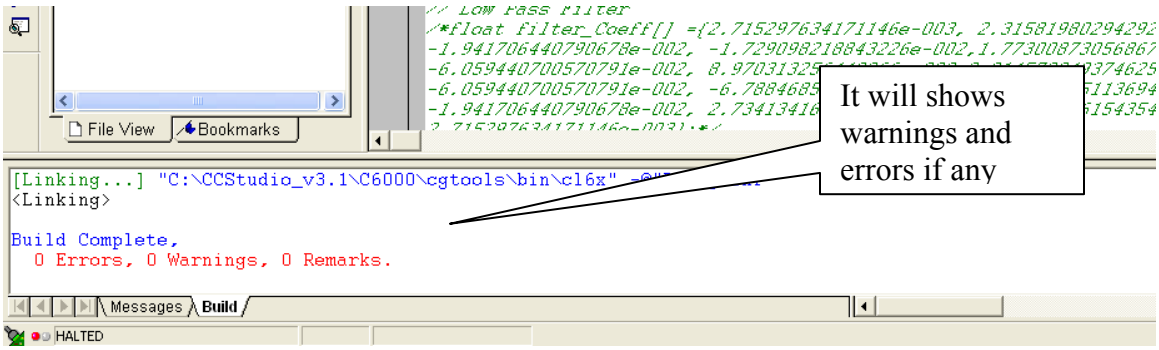


9 Go to Project-→ Build option → Preprocessor→set CHIP_6713 as shown below.

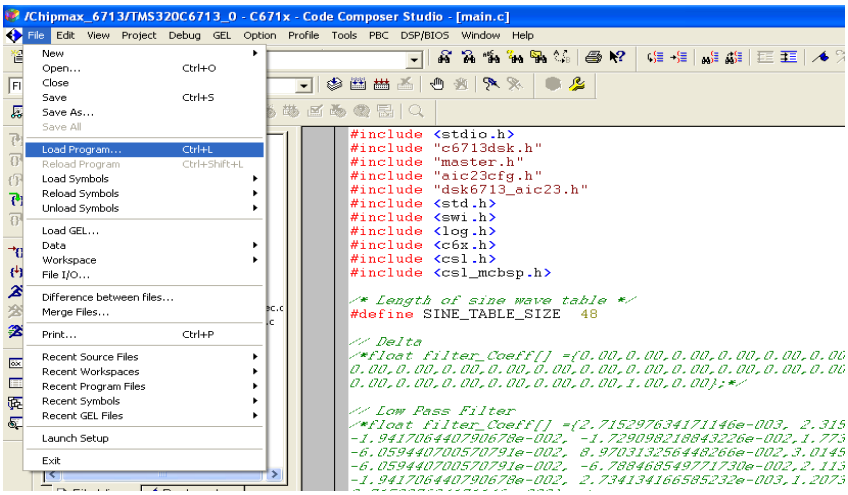


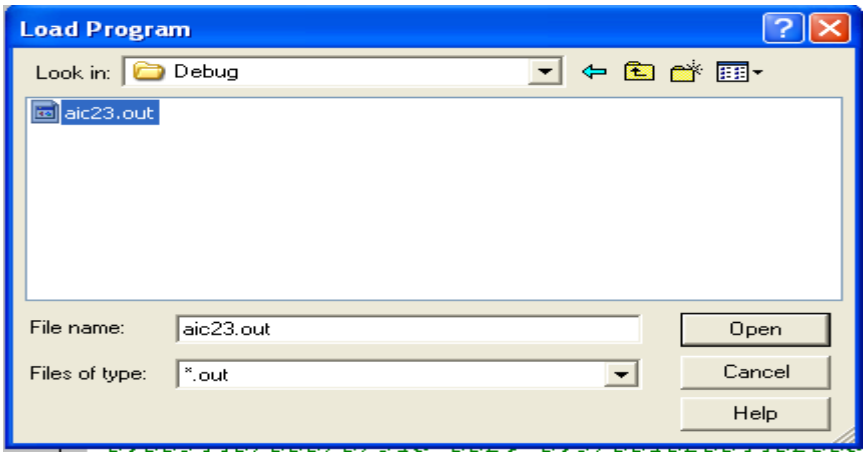
10. a) Go to Project to Compile.
b) Go to Project to Build.
c) Go to Project to Rebuild All.



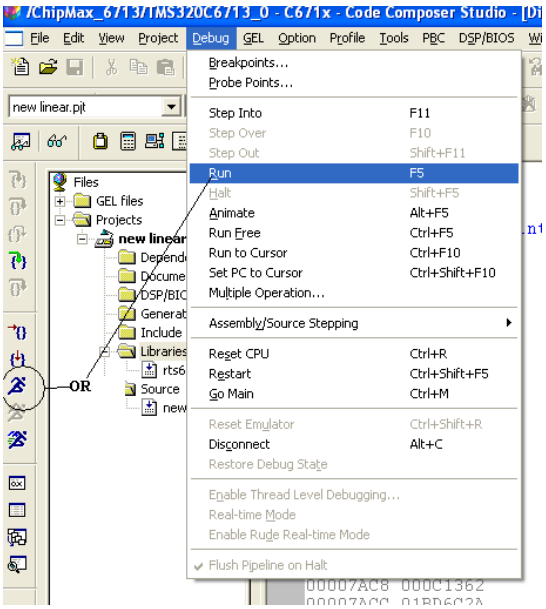


11. Put a break point at LoggerIndex = 0;
12. Go to file and load program and load **".out"** file into the board.

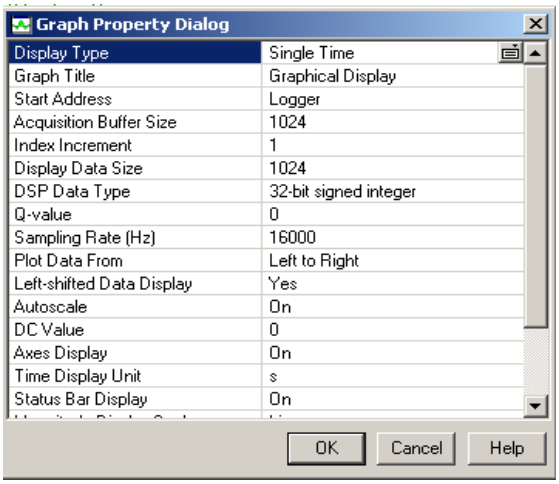


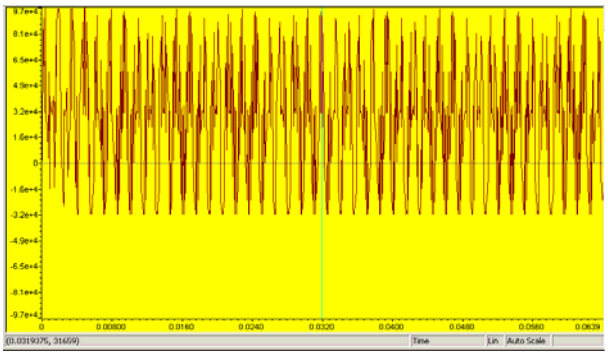


13. Go to Debug and click on run to run the program.

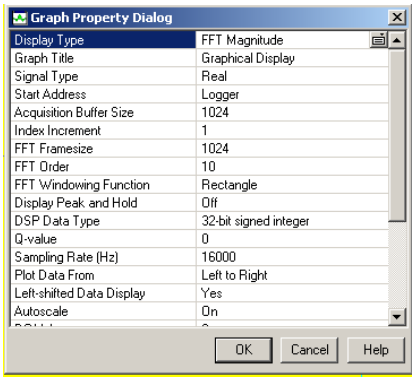


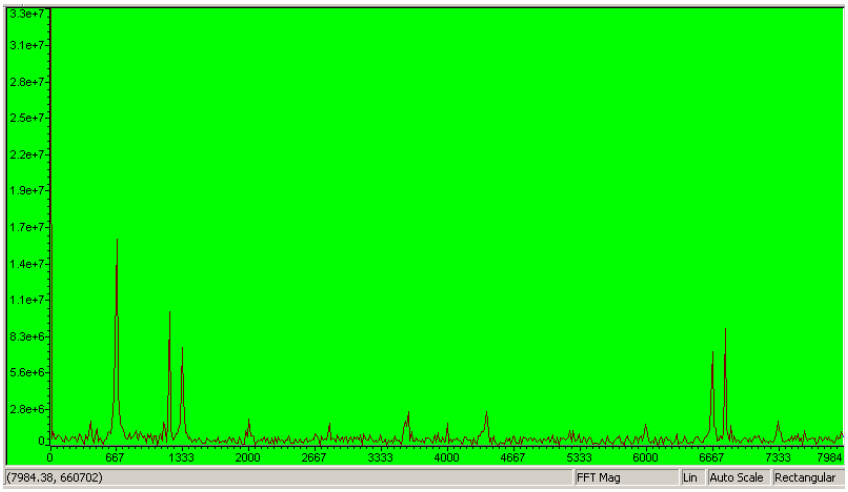
14. To see the Graph, go to View and select time/frequency in the Graph, and give the correct Start address provided in the program, Display data can be taken as per user.





15. To view the Spectrum, go to View and select time/frequency in the Graph, and give the correct Start address provided in the program, Display data can be taken as per user.

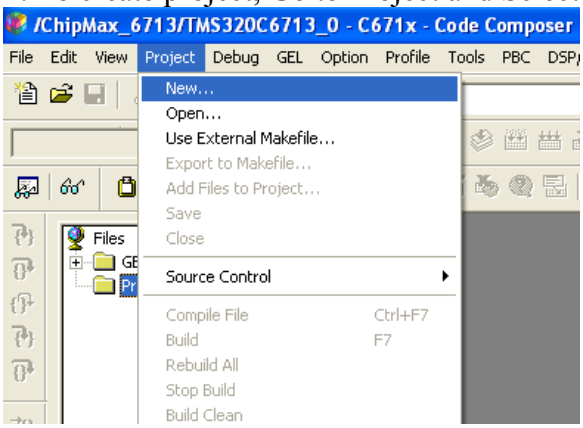




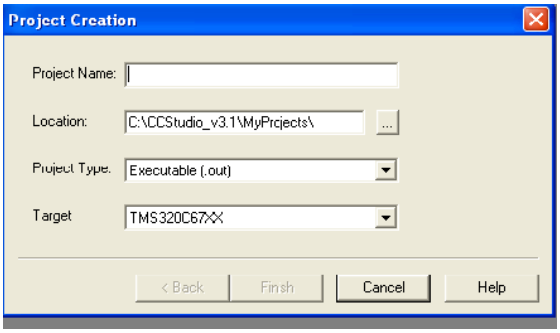
Experiment 6: Noise Removal

Procedure to create new Project:

1. To create project, Go to Project and Select New.

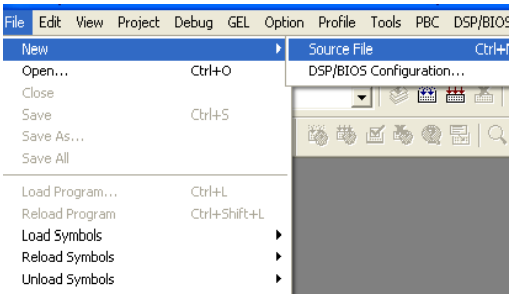


2. Give project name and click on finish.



(Note**:** Location must be c:\CCStudio_v3.3\MyProjects).

3. Click on File ⇨ New ⇨ Source File, to write the Source Code.



AIM: Noise removal in a given mixed signal.
Noise removal: In the real time systems every signal will get corrupted by the noise. To analyze the noise we need to add the noise, a noise will be generated and added to the signal. A noise of 3KHz is added to a tone of 400KHz then the noise is removed by using an high pass filter.

Program:

```
#include <stdio.h>
#include "c6713dsk.h"
#include "master.h"
#include "aic23cfg.h"
#include "dsk6713_aic23.h"

#define FiltLen 47
#define SINE_TABLE_SIZE  48

float PrNoise
[600]={0.3509,0.1118,0.3631,0.2673,0.2826,0.1909,0.2901,0.1378,0.1270,0.1798,
0.2925,0.3730,0.1307,0.1380,0.2859,0.1738,0.3278,0.0355,0.3177,0.2905,
0.1473,0.2380,0.2112,0.2662,0.3228,0.0264,0.2991,0.3027,0.1220,0.1676,
0.2360,0.2942,0.3058,0.3013,0.1423,0.1478,0.3479,0.2017,0.3278,0.1164,
0.2747,0.2752,0.4239,0.0666,0.2130,0.2124,0.0124,0.3306,0.3134,0.1420,
0.2837,0.0446,0.2438,0.1069,0.1902,0.3497,0.0962,0.1404,0.2046,0.1419,
0.3231,0.3962,0.0549,0.0875,0.2742,0.1418,0.3797,0.3756,0.2441,0.3271,
0.1456,0.2332,0.3789,0.2250,0.3798,0.3153,0.2295,0.2950,0.3924,0.0982,
0.1493,0.3649,0.1159,0.2095,0.3088,0.3656,0.2539,0.0606,0.0548,0.2847,
0.2610,0.4005,0.2985,0.2459,0.1789,0.0824,0.3774,0.2256,0.0309,0.2949,
0.2133,0.0136,0.3288,0.0923,0.0491,0.0178,0.1205,0.1107,0.3042,0.0761,
0.1512,0.1302,0.3434,0.1384,0.2454,0.1873,0.2804,0.0407,0.4164,0.3858,
0.0678,0.2275,0.1430,0.2304,0.0500,0.1793,0.1887,0.2067,0.1776,0.0374,
0.2449,0.2093,0.0772,0.1982,0.1787,0.2021,0.2916,0.2557,0.3304,0.0184,
0.0892,0.1823,0.3762,0.2984,0.1576,0.1120,0.0088,0.3026,0.2274,0.1223,
0.2151,0.1131,0.3248,0.1441,0.3475,-0.0471,0.2749,0.1925,0.1708,0.2431,
0.2975,0.0634,0.3891,0.0372,0.1486,0.0943,0.3835,0.0355,0.2320,0.2466,
0.2428,0.3175,-0.0105,0.2891,0.1764,0.2621,0.0814,0.2239,0.3074,0.4196,
0.2065,0.0613,0.0321,0.2495,0.3044,0.2513,0.1193,0.3635,0.2031,0.3801,
0.0303,0.0733,0.3001,0.3507,0.2985,0.1186,0.0656,0.3612,0.3397,0.1294,
0.1102,0.1194,0.3025,0.2892,0.1845,0.1956,0.1073,0.2981,0.3682,0.2311,
0.2244,0.2099,0.0990,0.2220,0.2894,0.2813,0.2316,0.0653,0.2872,0.1048,
0.1335,0.1639,0.1335,0.2752,0.0262,0.2958,0.2226,0.2916,0.1142,0.2017,
0.3252,0.2093,0.1280,0.4335,0.0627,0.1850,0.4388,0.1821,0.1451,0.0544,
0.3462,0.0154,0.0822,0.3031,0.1478,0.2130,0.2230,0.2897,0.0397,0.2436,
0.0428,0.3551,0.1458,0.3382,0.0957,0.2303,0.3804,0.0262,0.0356,0.0130,
0.2607,0.2102,-0.0020,0.2581,0.1933,-0.0047,0.0244,0.0922,0.3805,0.1092,
0.1670,0.1447,0.0477,0.3077,0.2124,0.2124,0.1879,0.1623,0.1219,0.2904,
```

0.0953,0.1132,0.0502,0.4002,0.3765,0.0096,0.1440,0.2291,0.1027,0.3114,
0.2580,0.2089,0.1434,0.3168,0.3503,0.2551,0.1064,0.3127,0.0588,0.1926,
0.3867,0.1418,0.1538,0.4241,0.0659,0.1438,0.2286,0.0508,0.2072,0.2740,
0.1688,0.2416,0.0827,0.2314,0.3496,0.1934,0.3886,0.0894,0.0704,0.1493,
0.1843,0.1859,0.1252,0.2559,0.0035,0.1217,0.0717,0.0912,0.0251,0.2405,
0.1436,0.1074,0.0493,0.1552,0.2456,0.3565,-0.0167,0.3025,-0.0187,0.2772,
0.2816,0.3582,0.0750,0.2422,0.2169,0.1210,0.2634,0.2424,0.0600,0.1631,
0.3293,0.1383,0.2371,0.1551,0.0338,0.1593,0.3720,0.1812,0.0607,0.1999,
0.1206,0.2411,0.1635,0.2727,0.2958,0.0758,0.0701,0.3565,0.2880,0.0363,
0.3726,0.2002,0.2003,0.2127,0.2768,0.3146,0.1400,0.0291,0.2936,0.1341,
0.3375,0.1544,0.3321,0.0716,0.0532,0.3473,0.0176,0.2671,0.2772,0.1617,
0.1264,0.3688,0.1475,0.1768,0.0764,0.1556,0.1539,0.3684,0.0979,0.1012,
0.1261,0.2401,0.0153,0.3234,0.0373,0.2052,0.0465,0.3405,0.3056,0.0901,
0.2585,0.1746,0.2905,0.1795,0.3366,0.1744,0.1618,0.2372,0.1421,0.3937,
0.1927,0.0760,0.1248,0.2367,0.1159,0.0431,0.3350,0.2452,0.1800,0.1234,
0.1133,0.2164,0.1742,0.2783,0.0053,0.3180,0.2518,0.0386,0.3270,0.0609,
0.2273,0.2060,0.0477,0.0029,0.3182,0.3218,0.1980,0.0483,0.2532,0.0704,
0.2688,0.1805,0.0634,0.3304,0.2816,0.3470,0.0396,0.0822,0.3077,0.2812,
0.2906,0.1659,0.1466,0.2278,0.3560,0.3095,0.2671,0.1798,0.3339,0.1612,
0.1967,0.2755,0.2225,0.2483,0.3013,0.2941,0.0201,0.0807,0.1395,0.2114,
0.0911,0.2137,0.1130,0.1435,0.2817,0.0310,0.0678,0.3107,0.1429,0.0814,
0.2429,0.3712,0.1587,0.2622,0.2169,0.1503,0.1834,0.3897,0.0846,0.1703,
0.2341,0.2631,0.3412,0.1344,0.0903,0.1993,0.0939,0.2847,0.1697,0.2312,
0.2761,0.1753,0.1190,0.0868,0.3864,0.0813,0.3023,0.2145,0.2764,0.3148,
0.0086,0.1329,0.3952,0.2684,0.3843,0.2181,0.0226,0.2681,0.1080,0.2249,
0.2832,0.2395,0.2193,0.1590,0.0663,0.1474,-0.0163,0.2260,0.3727,0.0303,
0.0235,0.3627,0.1062,0.0084,0.3246,0.0568,0.3485,0.0407,0.1353,0.2338,
0.2315,0.2545,0.2482,0.1510,0.0363,0.2882,0.0925,0.2603,0.3444,0.0445,
0.4034,0.0162,0.3319,0.3212,0.0066,0.2010,0.3604,0.1120,0.3155,0.2390,
0.3231,0.3301,0.0269,0.1138,0.2520,0.1875,0.3778,0.2939,0.2133,0.1170,
0.0977,0.3524,0.1664,0.3211,0.0252,0.3361,0.0023,0.3513,0.3686,0.2151,
0.0581,0.0777,0.1664,0.3211,0.0252,0.3361,0.0023,0.3513,0.3686,0.2151};

Note: // Generate the Noise from the Noise table from the matlab using rand() function//

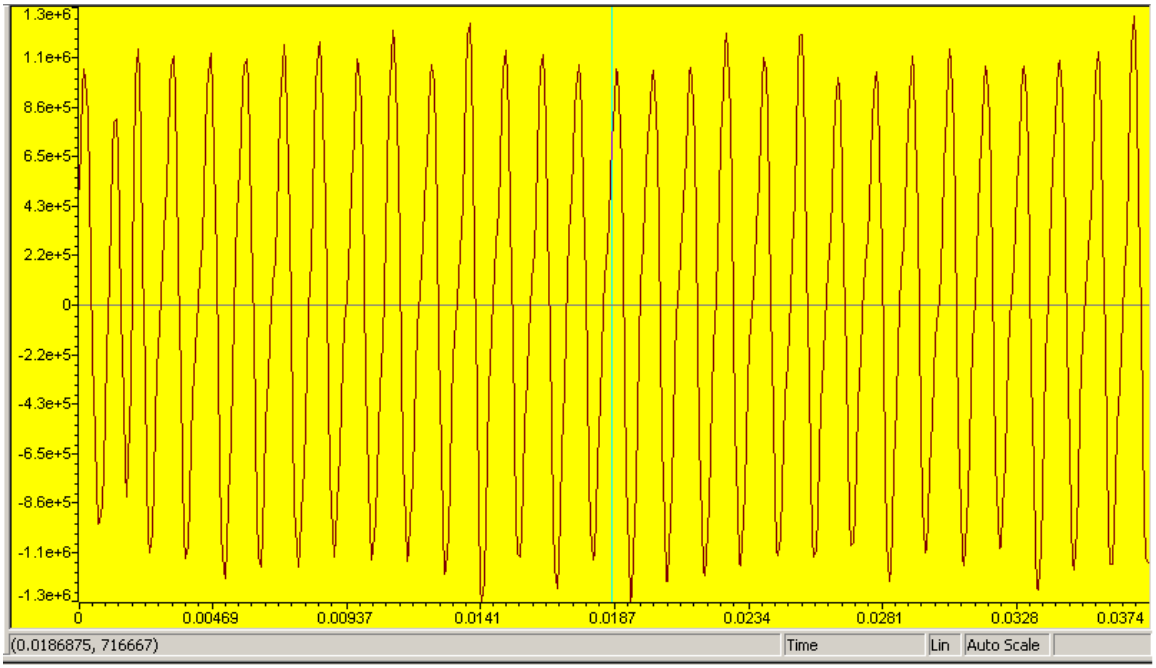
float RemNoise[47]={ -2.181030629062908e-002,8.917428211564256e-
003,1.182863130963947e-002, 1.428453375748106e-002,1.403350814552515^{e},
002,1.039856067649067e-002,5.093274200275827e-003, 1.421581262318843e-
003,2.515430936577368e-003,9.363053213111126e-003,1.949152112446623e-002,
2.722897698684972e-002,2.579943237700517e-002,1.050645191333675e-002,
1.832163998604397e-002, -5.406194181033640e-002,-8.514869723584616e-002,
9.887618777194764e-002,-8.633153232820758e-002, -4.651515024517261e
002,1.217011610629542e-002,7.394838432088444e-002,1.206802616409422e-001,
1.380624377729094e-001,1.206802616409422e-001,7.394838432088444e
002,1.217011610629542e-002, -4.651515024517261e-002,-8.633153232820758e-002,-
9.887618777194764e-002,-8.514869723584616e-002, -5.406194181033640e-002,-

```
1.832163998604397e-002,1.050645191333675e-002,2.579943237700517e-002,
2.722897698684972e-002,1.949152112446623e-002,9.363053213111126e
003,2.515430936577368e-003, 1.421581262318843e-003,5.093274200275827e
003,1.039856067649067e-002,1.403350814552515e-002, 1.428453375748106e-
002,1.182863130963947e-002,8.917428211564256e-003,-2.181030629062908e-002};

DSK6713_AIC23_Config config = { \
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
   0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
   0x00d8,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
   0x00d8,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */ \
   0x0010,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */      \
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */     \
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */             \
   0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
  0x008c,        /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */          \
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};

DSK6713_AIC23_CodecHandle hCodec;
Int32 InitWait =1;
Int32 data;

float in_buffer[100];
float InSigBuffer[600];
float NoiseBuffer[600];
float CorrSigBuffer[600];
float RecovSigBuffer[600];
int LogIndex =0;

main(){
        int i;
        LED=0x0;
        CSL_init();
        for( i = 0 ; i < 100; i++ ) in_buffer[i]=0.0;
        hCodec = DSK6713_AIC23_openCodec(0, &config);
        IRQ_globalEnable();
        IRQ_enable(IRQ_EVT_RINT1);
        IRQ_enable(IRQ_EVT_XINT1);
}

void led(){
        static int cc = 1;
        LED    = cc;
        if (cc == 0x03) cc = 0x05;
        else if (cc == 0x05) cc = 0x06;
```

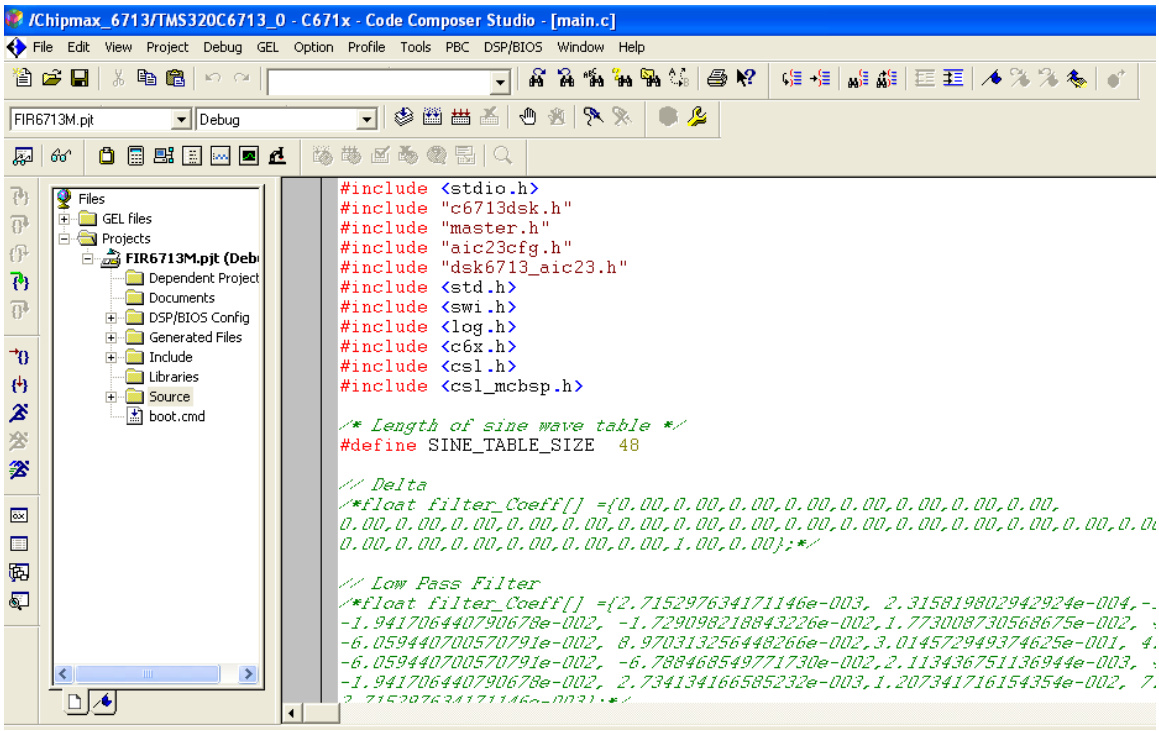
```
        else if (cc == 0x06) cc = 0x03;
        else cc = 0x03;
}
setpll200M(){
}
void read(){
        int i = 0;
        float result;
        float InData;
        float InNoise;
        float InCorrup;
        static int NoiseIndex=0;

        data=MCBSP_read(DSK6713_AIC23_DATAHANDLE);
        if(data>=32768) data= data|0xffff0000;
        InData = (Int32) (data)/16;
        InData=data;
        InNoise = PrNoise[NoiseIndex++]*4096 - 1024;
        if (NoiseIndex == 600)
                NoiseIndex = 0;
        InCorrup = InData + InNoise;
        for( i = 0; i <= (FiltLen-2); i++) in_buffer[i] = in_buffer[i+1];
        in_buffer[FiltLen-1]=InCorrup;
        result = 0;
        for( i = 0 ; i <= (FiltLen-1); i++ ) result += RemNoise[i] * in_buffer[i];
        data = (Int32)(result*512);
        InSigBuffer[LogIndex]= InData;
        NoiseBuffer[LogIndex]= InNoise ;
        CorrSigBuffer[LogIndex]= InCorrup;
        RecovSigBuffer[LogIndex] = (float)data;
        LogIndex++;
        if (LogIndex == 600)
                LogIndex =0;
}

void write(){
        if (InitWait<1000){
                InitWait++;
                MCBSP_write(DSK6713_AIC23_DATAHANDLE, 0);
                MCBSP_write(DSK6713_AIC23_DATAHANDLE, 0);
        }
        else{
                MCBSP_write(DSK6713_AIC23_DATAHANDLE, data);
                MCBSP_write(DSK6713_AIC23_DATAHANDLE, data);
        }
}
```

Output:

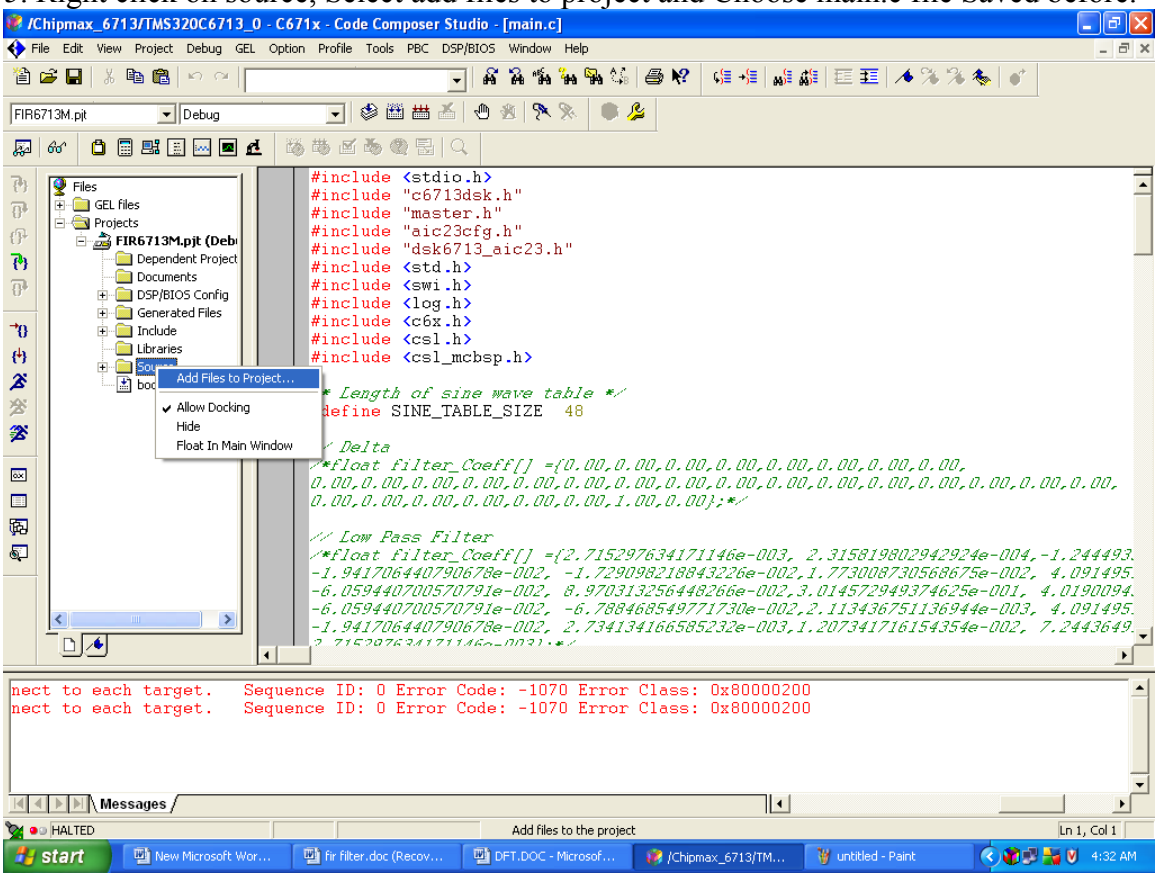
Recovered signal can be seen as shown below.



4. Enter the source code and save the file with main.c extension.

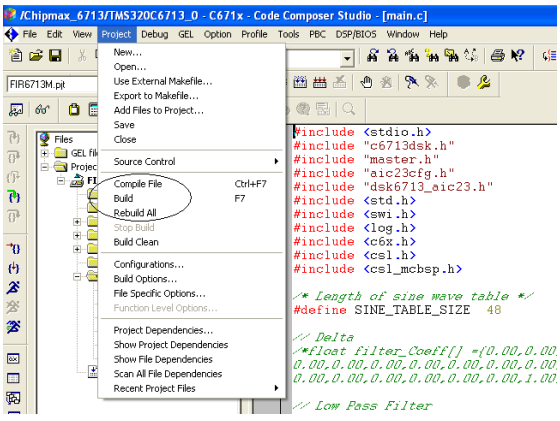


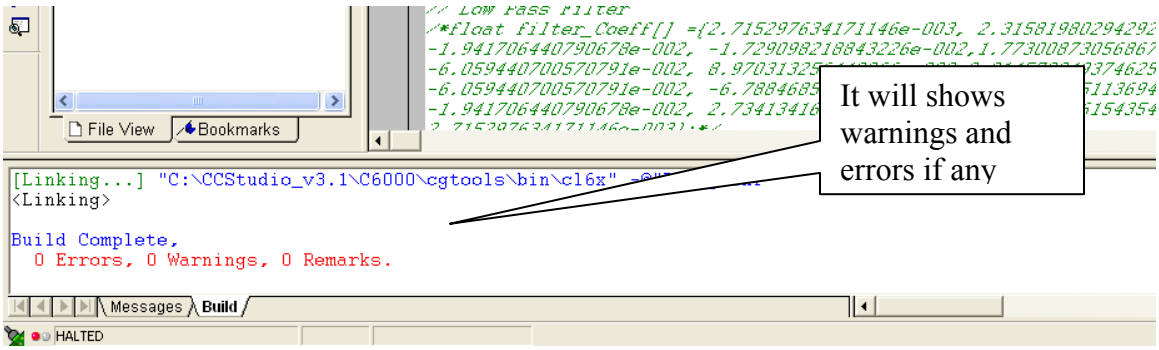
5. Right click on source, Select add files to project and Choose main.c file Saved before.



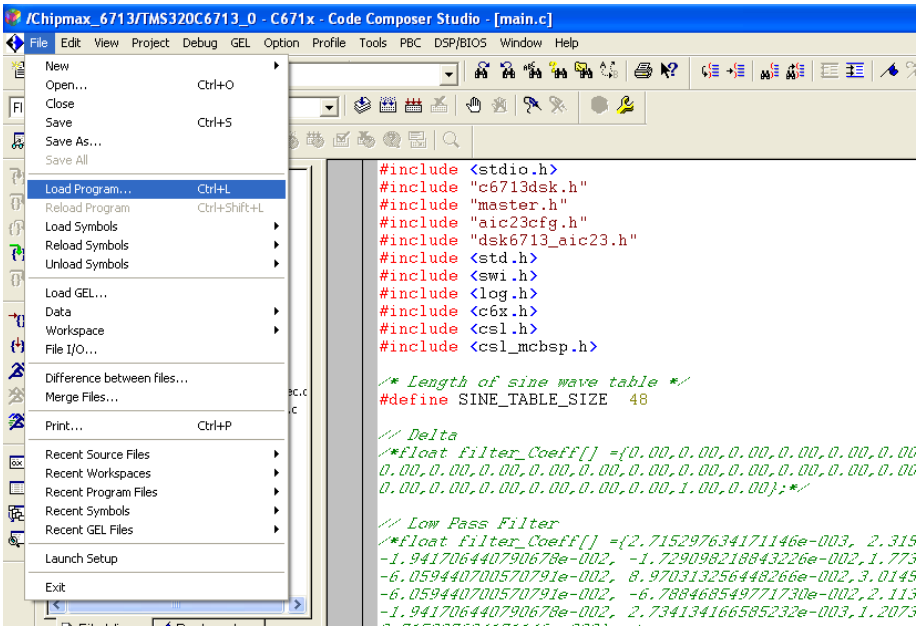
6. Add the other supporting .c /.asm/.cmd/.tcf files which configure the audio codec. Follow the same procedure which you have done in experiment 4 and 5.

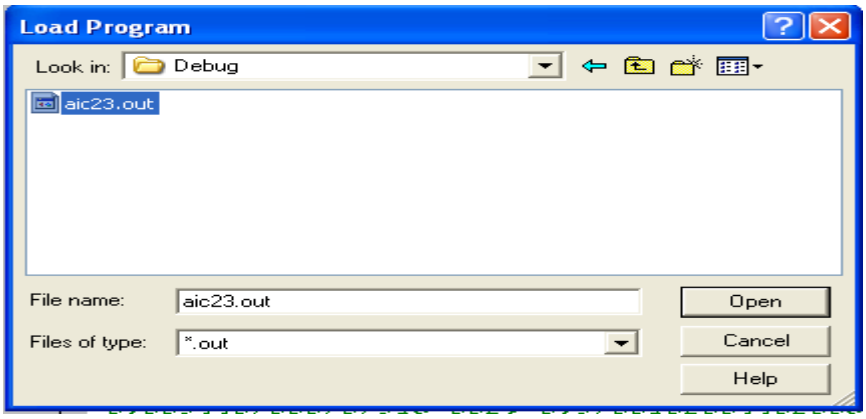
7. a) Go to Project to Compile.
   b) Go to Project to Build.
   c) Go to Project to Rebuild All.



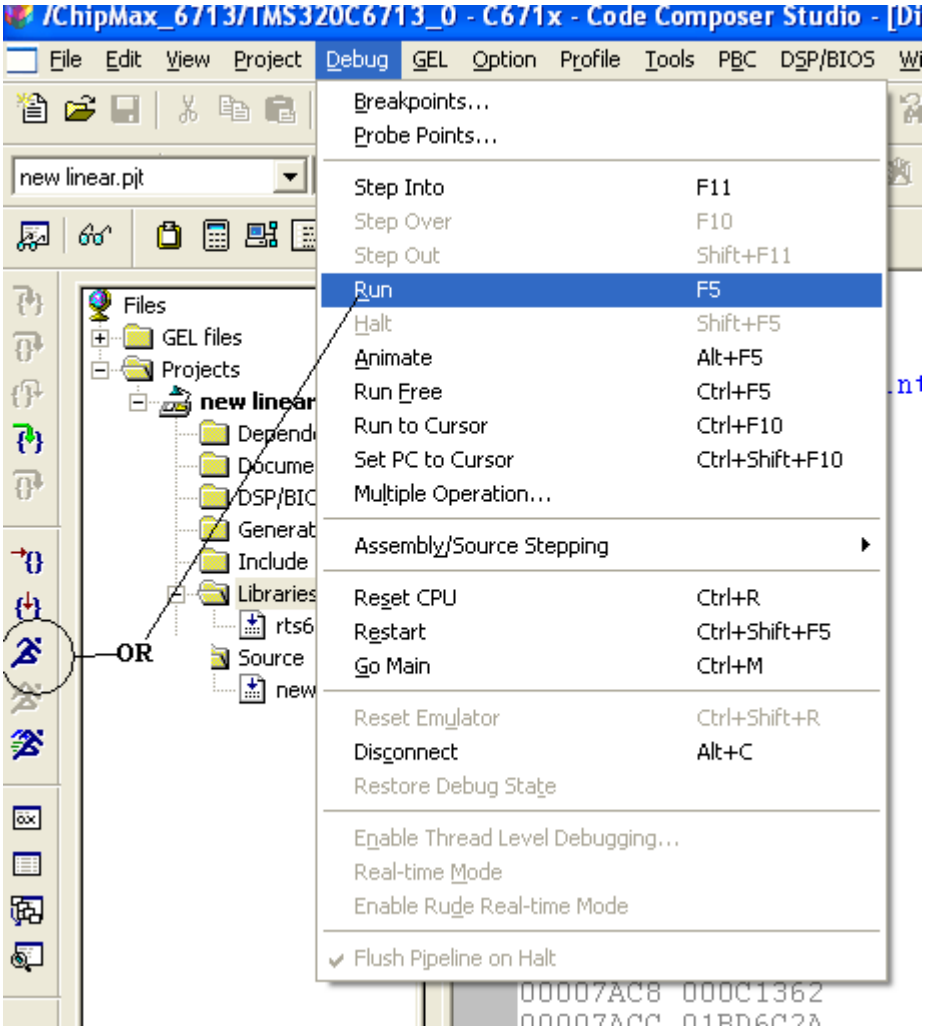


8. Put a break point at LogIndex = 0 in the source code.
9. Go to file and load program and load **".out"** file into the board.



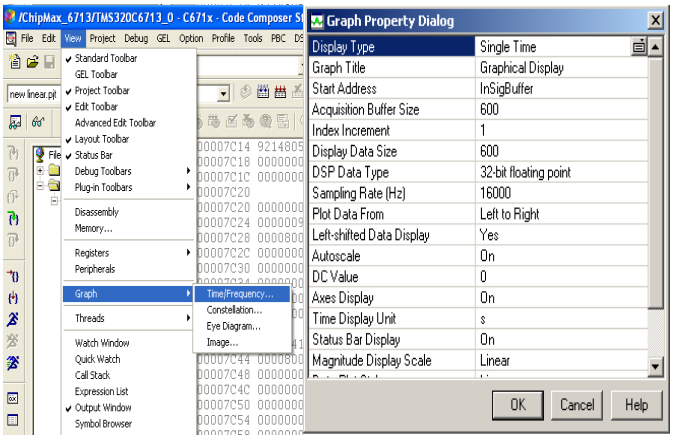


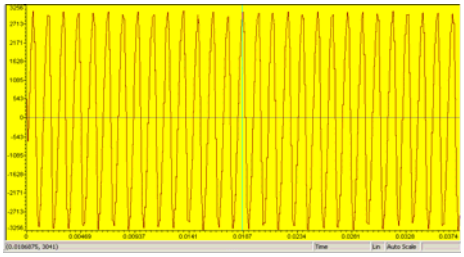
10. Go to Debug and click on run to run the program.



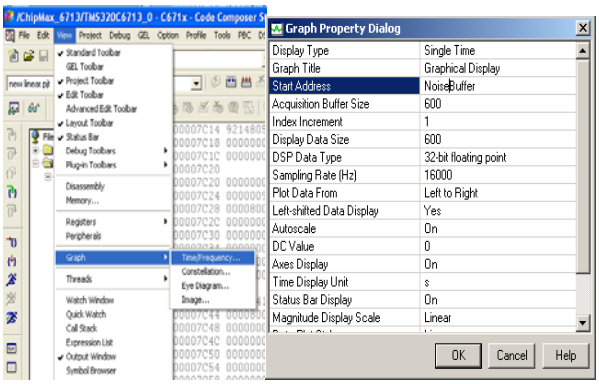
11. To see the Graph go to View and select time/frequency in the Graph and give the correct Start address provided in the program, Display data can be taken as per user.

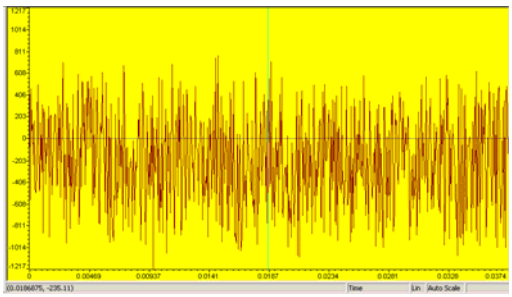
11.a) To plot the input signal:



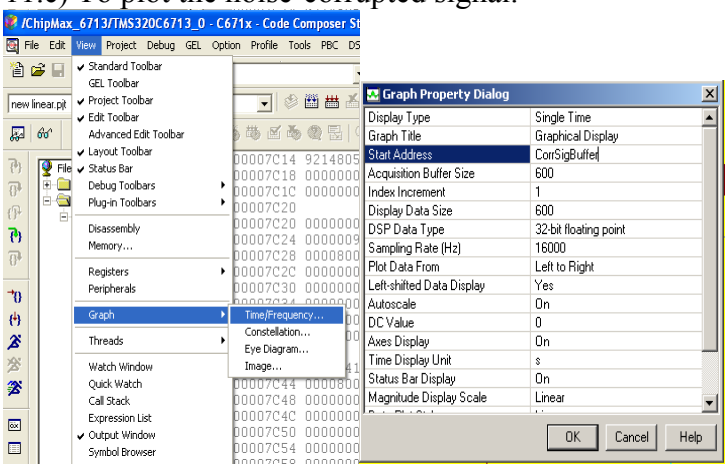


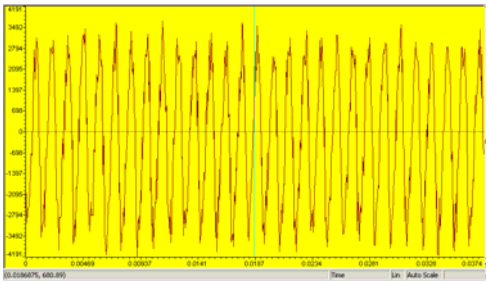
11.b) To plot the noise.



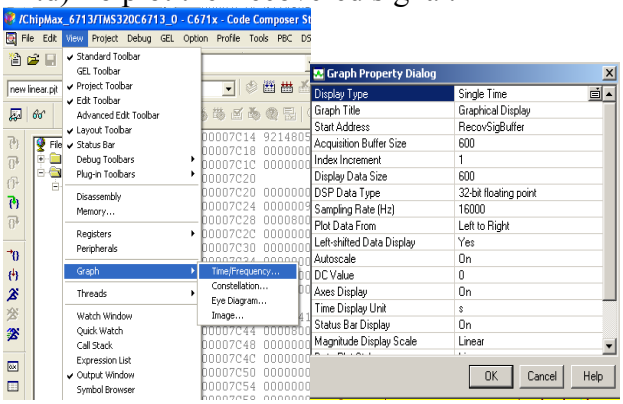


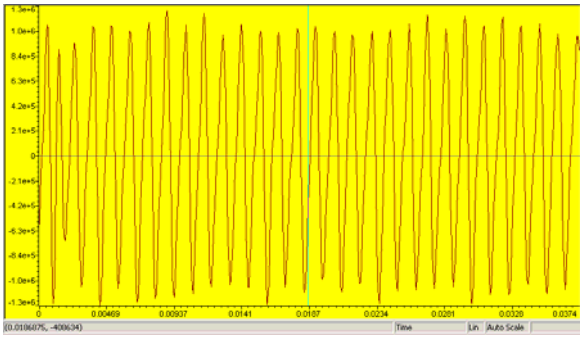
11.c) To plot the noise-corrupted signal.





11.d) To plot the Recovered signal.

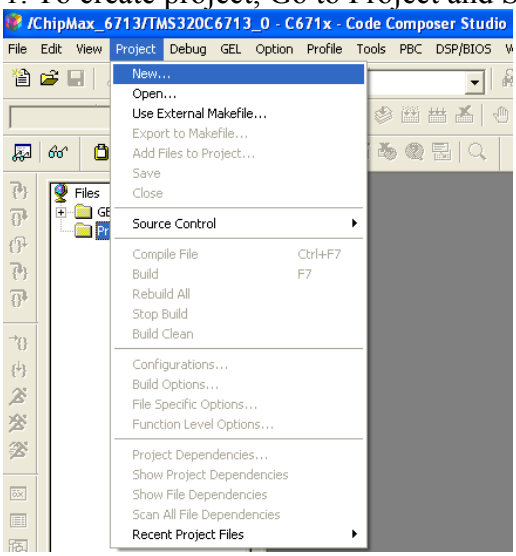




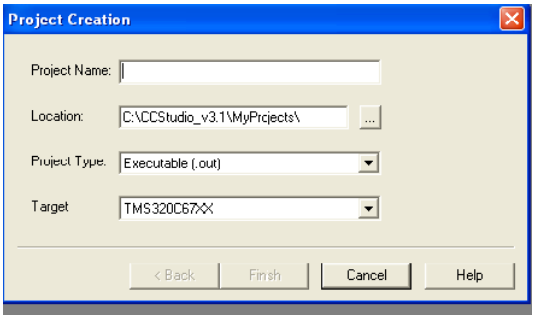
Experiment 7: Impulse Response

Procedure to create new Project:

1. To create project, Go to Project and Select New.

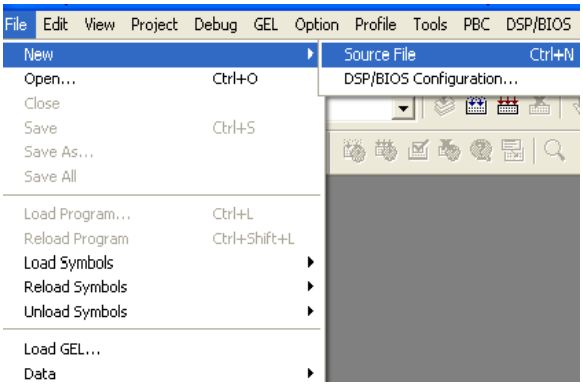


2. Give project name and click on finish.



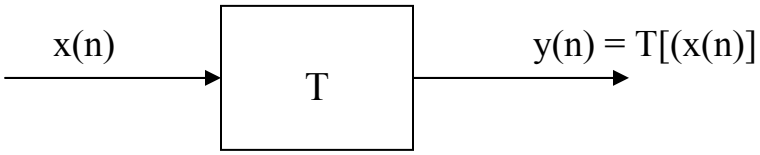
(**Note:** Location must be c:\CCStudio_v3.3\MyProjects).

3. Click on File ⇨ New ⇨ Source File, To write the Source Code.



AIM: To find Impulse response of a first order and second order system.

A discrete time system performs an operation on an input signal based on predefined criteria to produce a modified output signal. The input signal $x(n)$ is the system excitation, and $y(n)$ is the system response. The transform operation is shown as,



The convolution sum can be represented by, $y(n) = x(n) * h(n)$

For Example let's find out an impulse response of a difference equation.
The general form of difference equation is,

$$y(n) = \sum_{k=1}^{M} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k)$$

Find out the impulse response of second order difference equation.

Program:

```
#include<stdio.h>
#define Order  2
#define Len    5
float h[Len] = {0.0,0.0,0.0,0.0,0.0},sum;

void main()
{
int j, k;
float a[Order+1] = {0.1311, 0.2622, 0.1311};
float b[Order+1] = {1, -0.7478, 0.2722};

for(j=0; j<Len; j++)
   {
    sum = 0.0;
    for(k=1; k<=Order; k++)
      {
       if ((j-k) >= 0)
            sum = sum+(b[k]*h[j-k]);
      }
if (j <= Order)
```

```
    h[j] = a[j]-sum;
else
    h[j] = -sum;
printf (" %f     ",h[j]);
    }
}
```

Output:

0.131100 0.360237 0.364799 0.174741 0.031373

For first order difference equation.

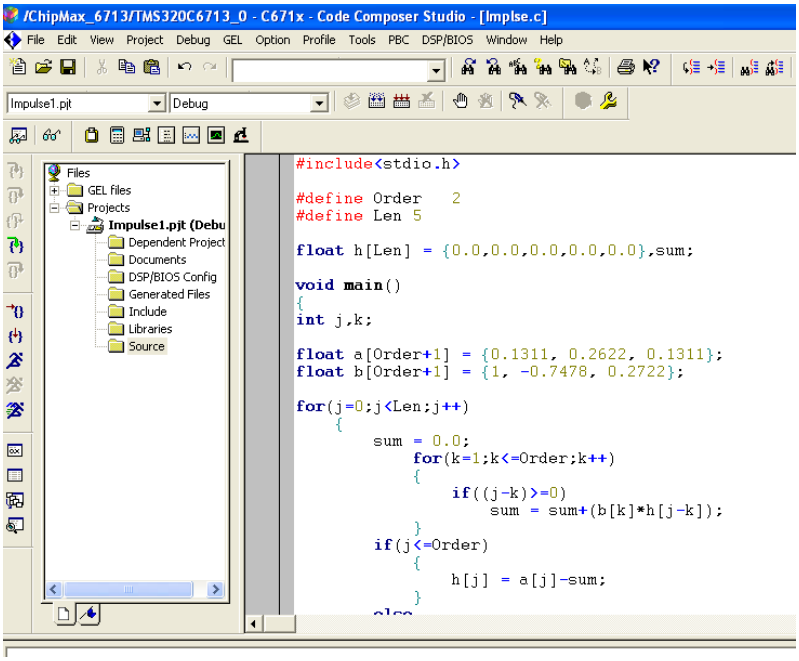
Program:

```
#include<stdio.h>
#define Order  1
#define Len    5
float h[Len] = {0.0,0.0,0.0,0.0,0.0},sum;

void main()
{
int j, k;
float a[Order+1] = {0.1311, 0.2622};
float b[Order+1] = {1, -0.7478};
for(j=0; j<Len; j++)
   {
    sum = 0.0;
    for(k=1; k<=Order; k++)
        {
        if((j-k)>=0)
        sum = sum+(b[k]*h[j-k]);
        }
if(j<=Order)
     h[j] = a[j]-sum;
else
     h[j] = -sum;
printf("%f       ",h[j]);
   }
}
```

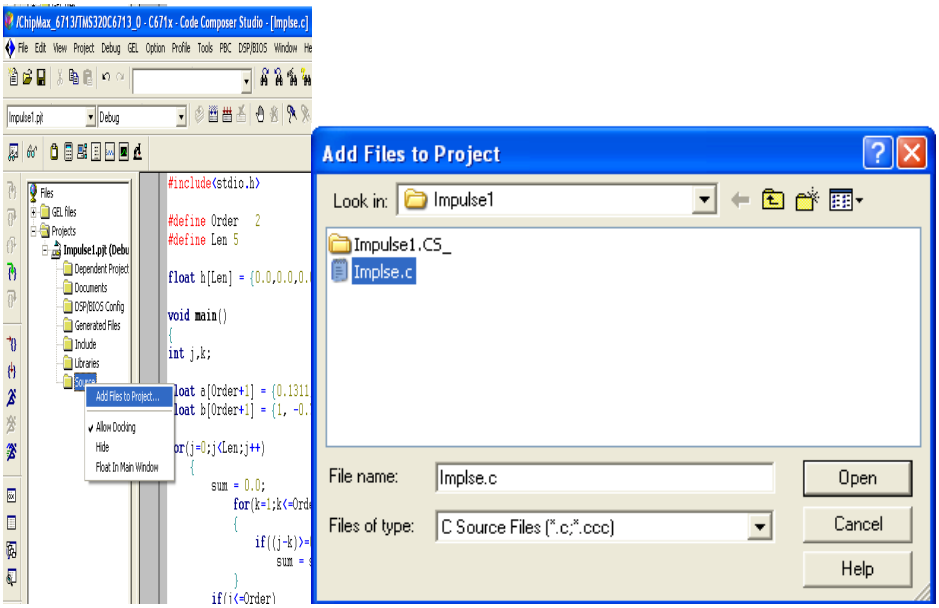
Output:

0.131100 0.360237 0.269385 0.201446 0.150641

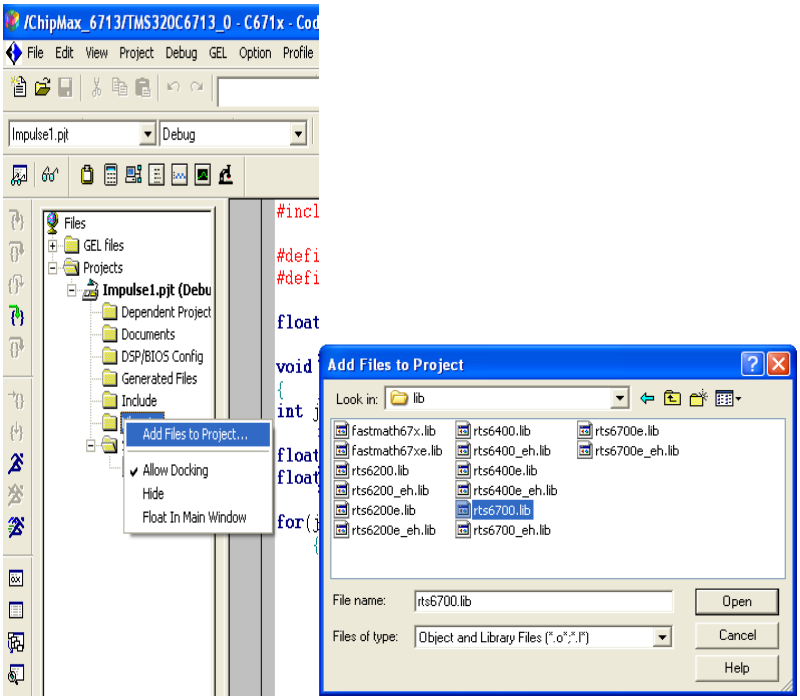
4. Enter the source code and save the file with "**.C**" extension.



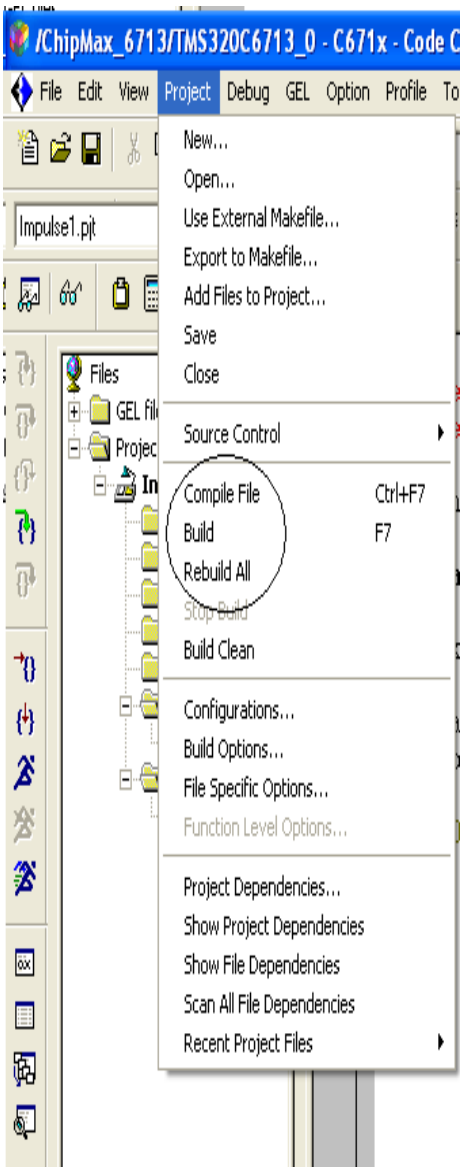
5. Right click on source, Select add files to project. And Choose "**.C** " file Saved before.

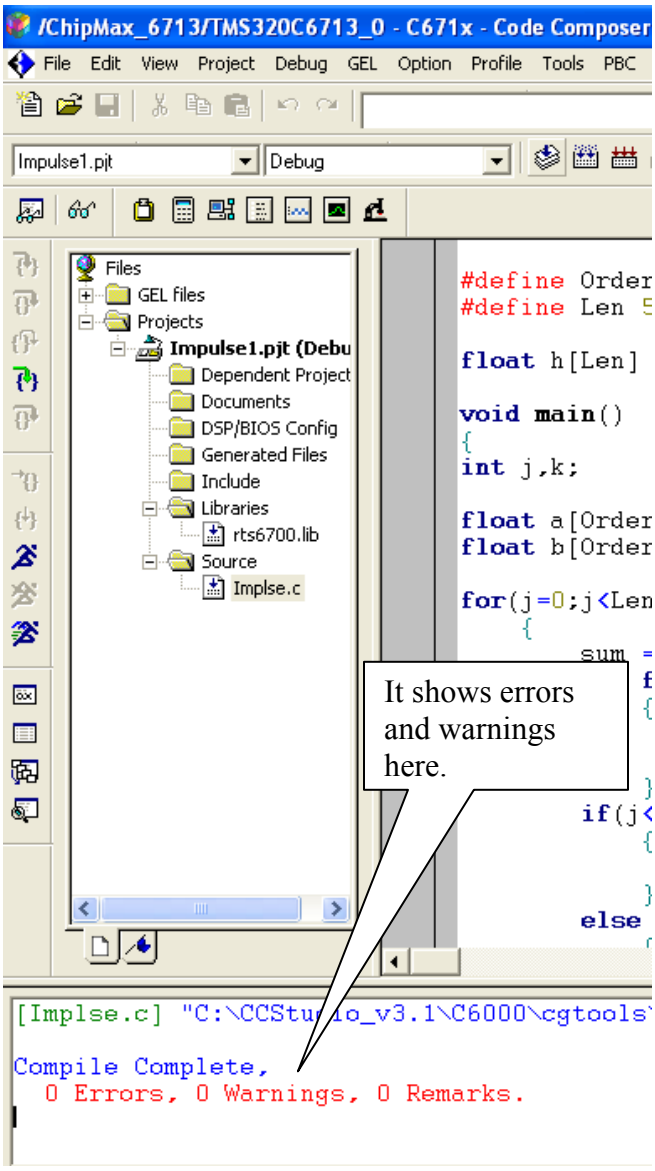


6. Right Click on libraries and select add files to Project.. and choose C:\CCStudio_v3.3\C6000\cgtools\lib\rts6700.lib and click open.

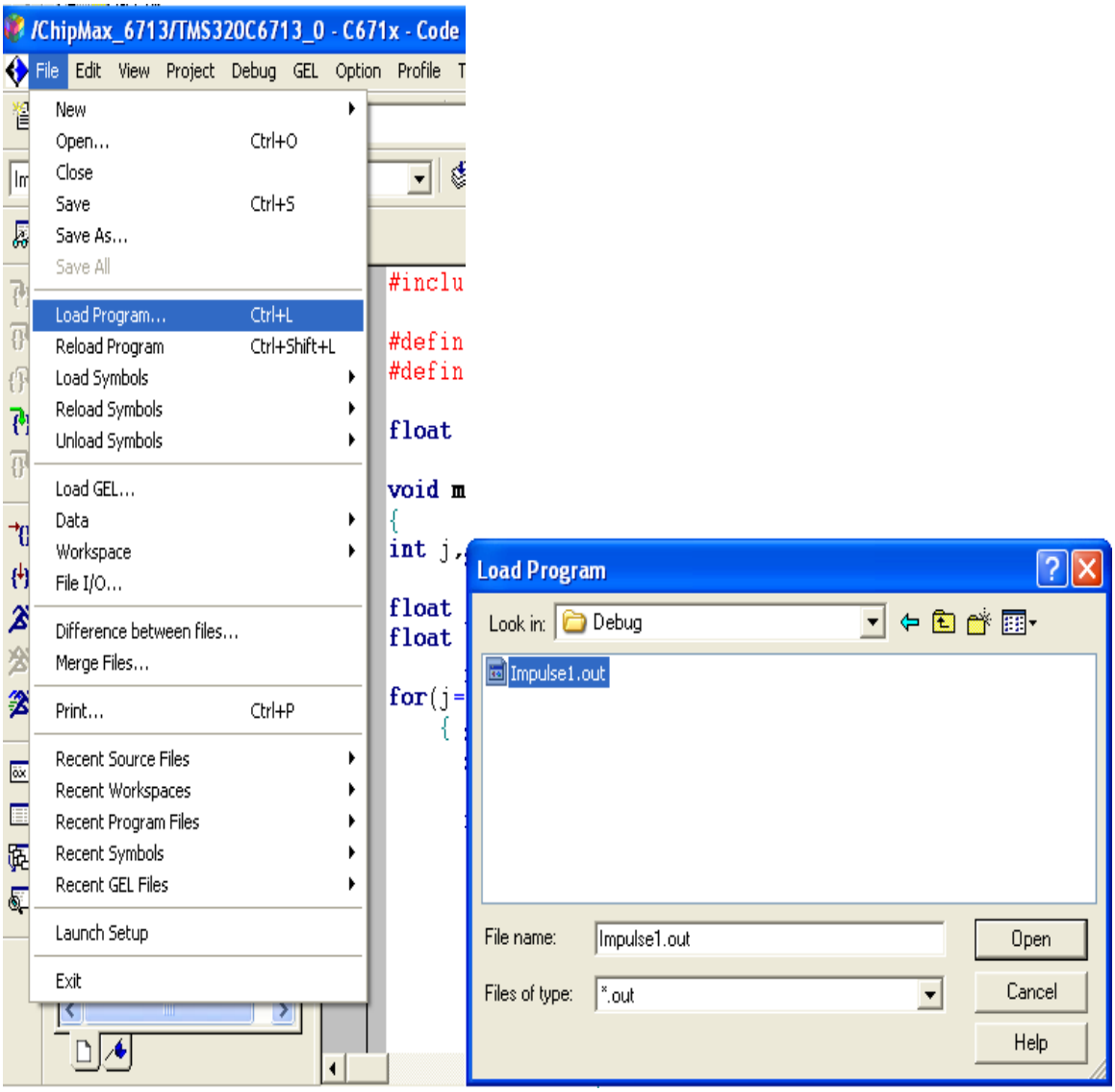


7. a) Go to Project to Compile.
   b) Go to Project to Build.
   c) Go to Project to Rebuild All.

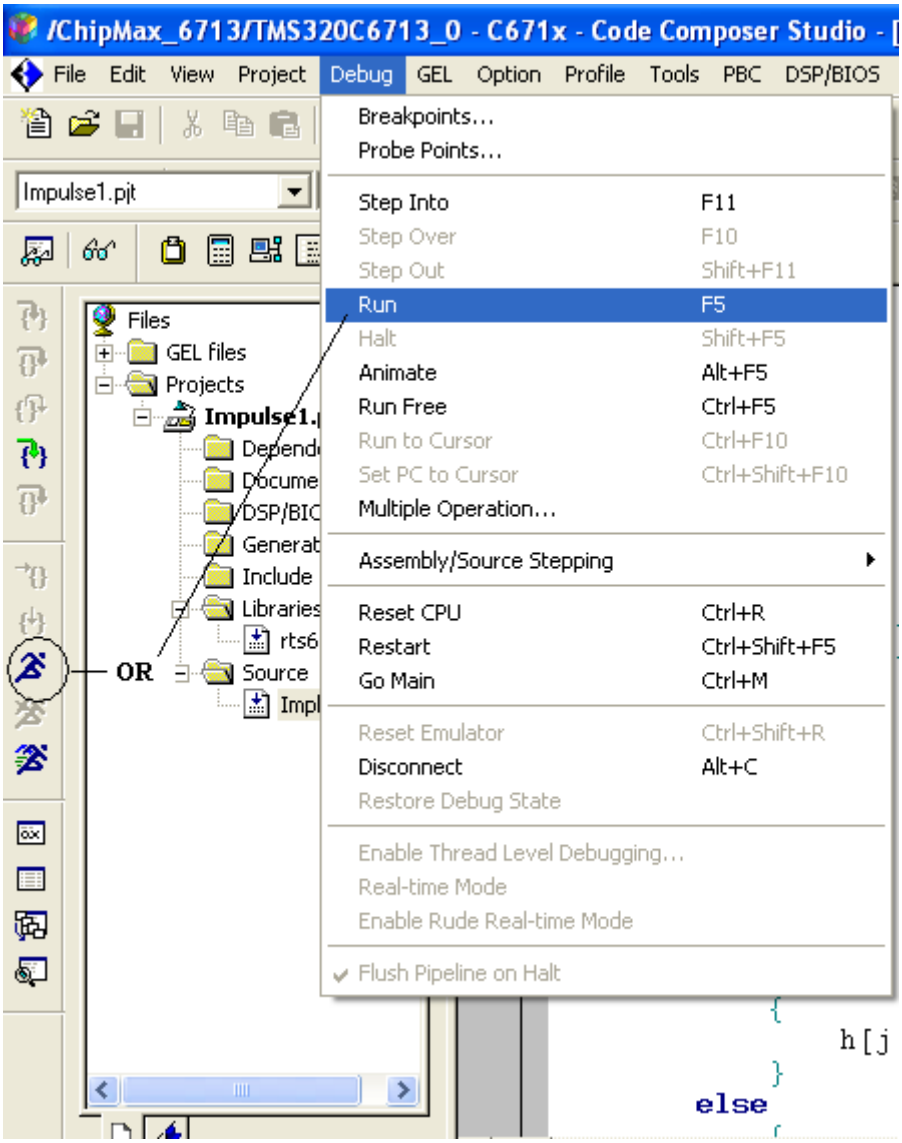




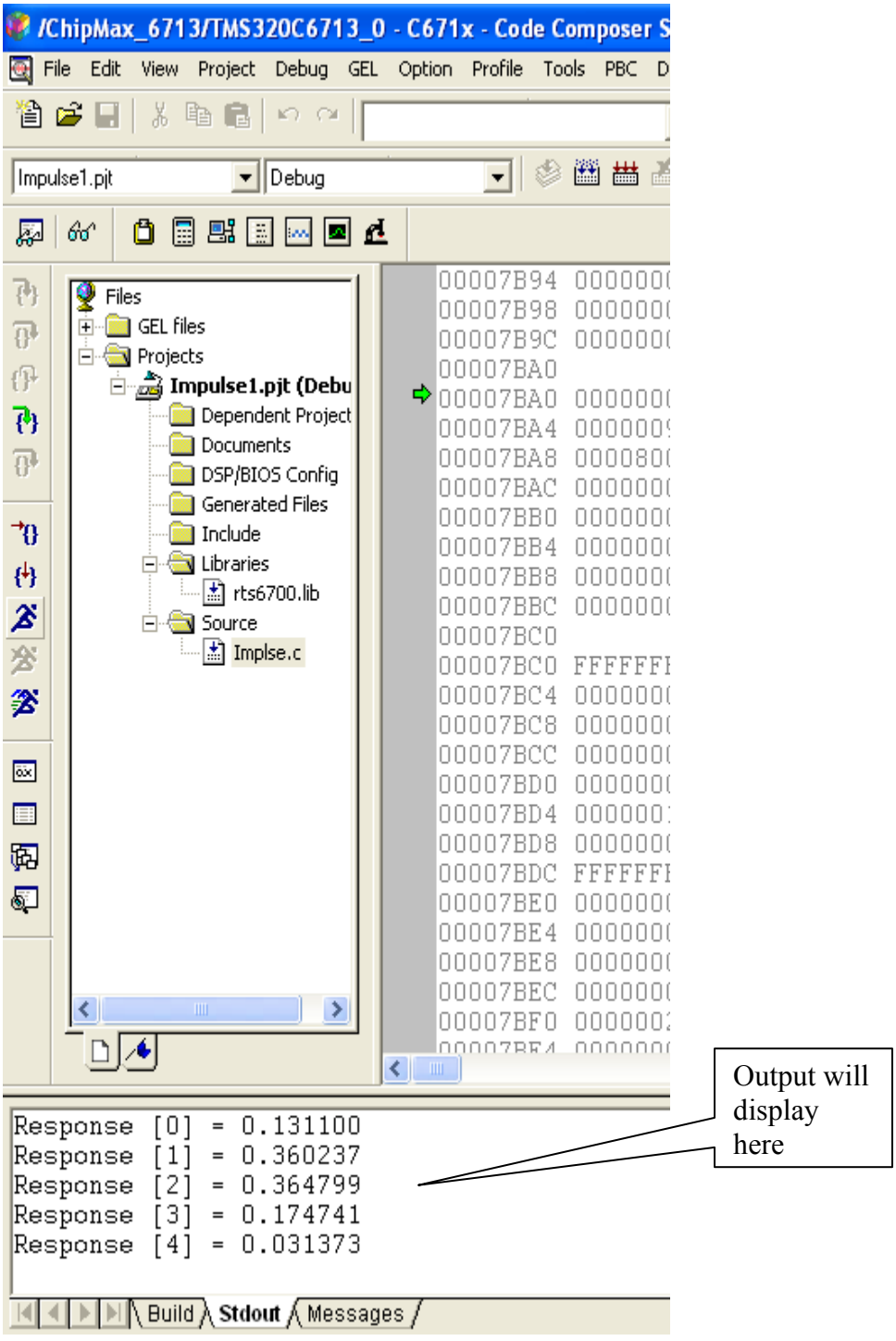
8. Go to file and load program and load **“.out”** file into the board..



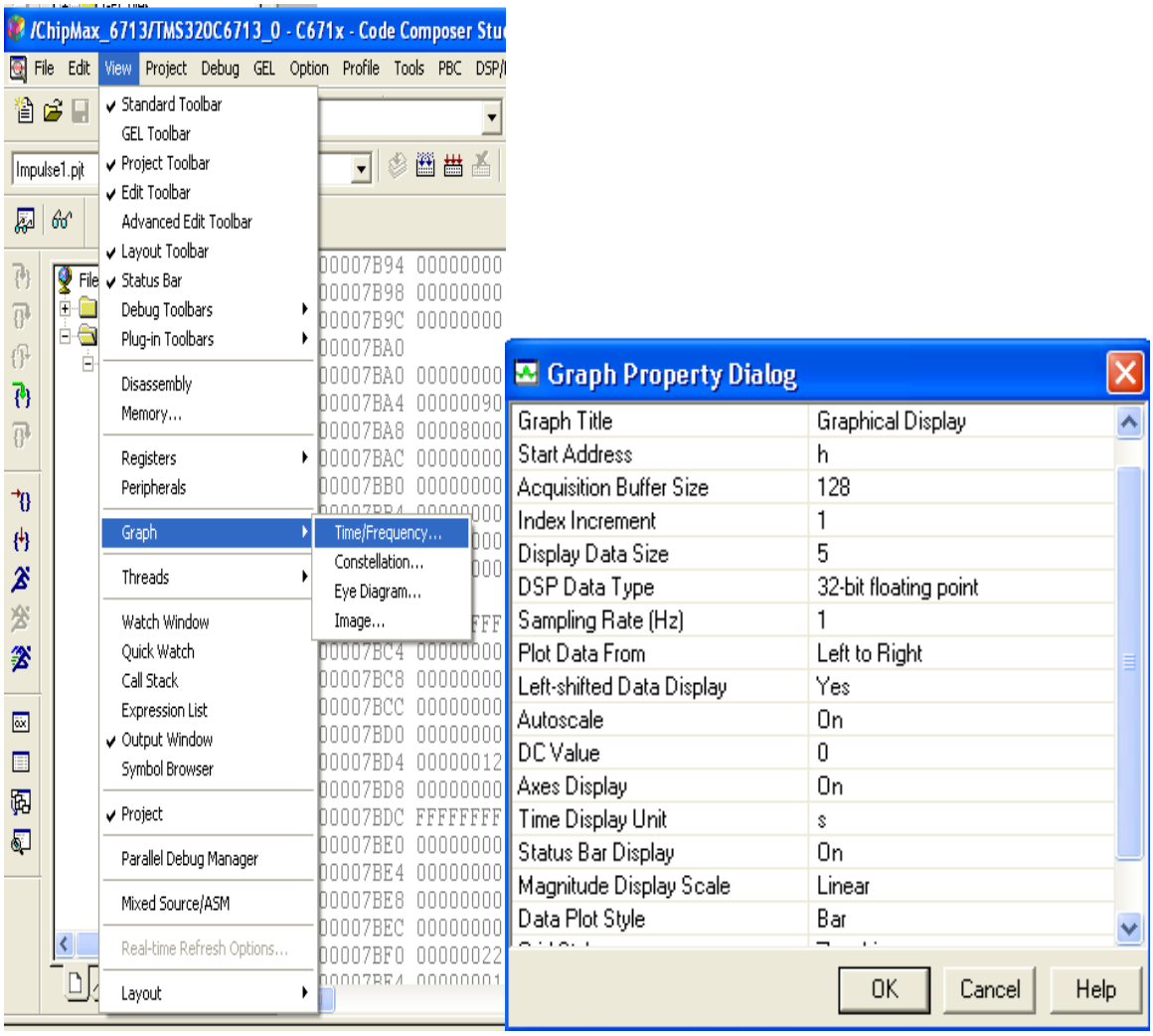
9. Go to Debug and click on run to run the program.



10. Observe the output in output window.



11. To see the Graph go to View and select time/frequency in the Graph, and give the correct Start address or the output variable name provided in the program, Display data can be taken as per user. Select DSP data type as 32-bit floating point.



12. Green line is to choose the point, Value at the point can be seen (Highlighted by circle at the left corner).